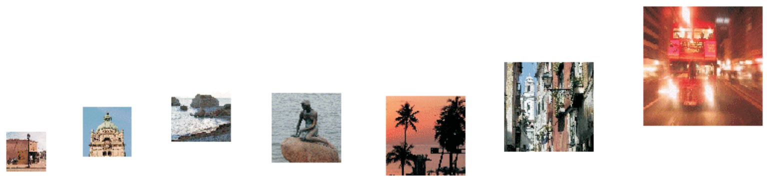
Francesca Toni   Paolo Torroni (ed.)

Computational Logic in Multi-Agent Systems

Sixth International Workshop

City University London, UK, 27-29 June, 2005

Proceedings



CLIMA VI Home Page:
`http://clima.deis.unibo.it/`

Preface

Multi-Agent Systems are communities of problem-solving entities that can perceive and act upon their environments to achieve their individual goals as well as joint goals. The work on such systems integrates many technologies and concepts in artificial intelligence and other areas of computing as well as other disciplines.

Agent-related concepts have recently increased their influence in the research and development of Computational Logic based systems. Computational Logic provides a plethora of well-defined, general, and rigorous frameworks for studying syntax, semantics and operational models for individual agents and multi-agent systems, for attending implementations, environments, tools, and standards, and for linking together specification and verification of properties of individual agents and multi-agent systems.

The first workshop on Multi-Agent Systems in Logic Programming was held in Las Cruces, New Mexico, USA, in December 1999, in conjunction with the Sixteenth International Conference on Logic Programming, with the aim to explore the application of Logic Programming to Multi-Agent Systems. This workshop can be seen as a precursor of the First International Workshop on Computational Logic in Multi-Agent Systems (CLIMA), which was held in July 2000 at Imperial College London, UK, under the umbrella of CL'2000, co-located with other major Computational Logic events. Ever since, CLIMA has been a forum to discuss techniques for representing, programming, and reasoning formally about agents and multi-agent systems, by means of Computational Logic-based techniques.

CLIMA is now in its sixth edition. It received 30 submissions, of which only 16 were selected for presentation and inclusion in the present volume. Among the topics covered by this year's programme, we find agent reasoning and reasoning about knowledge and belief, social and normative aspects of multi-agent systems, coordination of multi-agent reasoning, planning, verification.

We feel that the important results achieved at the intersection of Computational Logic and Multi-Agent Systems have now reached a "critical mass" and, in order to start promoting their dissemination in a systematic way, CLIMA VI is also hosting a *tutorial programme*, covering a broad range of topics, from rational agent programming to normative and social reasoning and verification, and a number of formalisms, from temporal logics to logic programming to deontic logic and beyond. The tutorial programme is offered with the aims to introduce Computational Logic-based agent programming environments, to introduce novices to state-of-the-art Computational Logic-based research in Multi-Agent Systems, and to provide guidelines to researchers and practitioners interested in logic-based agent technologies. It includes the following topics:

- BDI agent programming in AgentSpeak, through an overview of the features available with the agent programming language *Jason*;
- *rational agent groups programming*, and how to implement individual agents through the direct execution of a formal description of individual (rational)

agent behaviour given, using a combination of temporal logic and logics concerning belief and ability;
– multi-agent systems programming based on the multi-threaded and distributed logic programming framework *Qu-Prolog*;
– representation of norms of behaviour and institutional aspects of (human or computer) societies, using the language $(\mathcal{C}/\mathcal{C}^+)^{++}$;
– agent specifications in abductive logic programming and logic programming with priorities, i.e., the $\mathcal{KGP}$ model of agency;
– theory and practice of design, specification, verification and testing of interaction protocols for open agent societies, using *SOCS*-$\mathcal{SI}$.

Fostering dissemination of results is important, but needs to be accompanied by establishing evaluation criteria that can serve as milestones for testing new approaches and techniques. To this end, similarly to what is being done in various parts of artificial intelligence (theorem proving, planning, operations research, constraint programming, robo-cup etc.) and, lately, also in specialised areas in agent systems (trading agents), we decided to promote a CLIMA contest, where original, innovative, and effective application of Computational Logic-based techniques can be confronted in solving specific multi-agent issues. CLIMA VI is thus also hosting the *First CLIMA Contest*, organised by Mehdi Dastani, from the University of Utrecht, and Jürgen Dix, from the Technical University of Clausthal. In order to render the aims of the contest concrete, we opted for a particular scenario that serves as a basis for such a contest. The formulation of such a scenario has turned out to be a very hard task. We hope that this initiative will be a first step towards the definition of paradigms and measuring standards that can further logic-based intelligent agent research and its application.

In addition to hosting the first CLIMA tutorial proramme and the first CLIMA contest, CLIMA VI is also collocated with the final *dissemination workshop of the SOCS project*, a 42 months EU-funded research project which involved 6 European academic partners and whose purpose was to produce a Computational Logic model for the description, analysis and verification of global and open societies of heterogeneous agents, called *computees* (standing for computational logic-based agents). The collocation of the two events is motivated by the research scope of SOCS being well within CLIMA's scope. The SOCS dissemination event will consist of 2 tutorials (on the $\mathcal{KGP}$ model and on the *SOCS*-$\mathcal{SI}$ framework, both outcomes of SOCS), and of a number of talks, aimed at presenting and discussing the main research challenges and achievements of the project. The talks will span a range of topics, from the motivations to adopt formal approaches to programming Multi-Agent Systems, to the kind of results to be expected from such research, from the problems of combining different Computational Logic frameworks into a single agent framework, to the issues of evaluating and testing rational agent systems.

Finally, we are very proud to introduce Robert Kowalski as the CLIMA VI keynote speaker. His talk will advocate the need of decision theoretic features in logic-based agents. It will also motivate the need for a pre-activity feature in agent systems, in addition to the conventional reactivity and pro-activity features.

This volume collects the proceedings of the CLIMA VI workshop, the final SOCS dissemination event, the first CLIMA tutorial programme and the first CLIMA contest. Therefore, besides the regular CLIMA papers and the abstract of the invited talk, it contains 5 short papers describing the 4 accepted contest entries and the contest itself, and the abstracts of the tutorials and of the talks of the SOCS dissemination event.

We hope that this rich programme will stimulate interesting discussions and will help promoting research in the area.

We would like to express our gratitude to the invited tutorialists: Rafael Bordini, from the University of Durham, Federico Chesani from the University of Bologna, Keith Clark, Silvana Zappacosta-Amboldi, Fariba Sadri, and Marek Sergot, all from Imperial College London, Michael Fisher, from the University of Liverpool, Marco Gavanelli, from the University of Ferrara, and Kostas Stathis, from City University London, and to the keynote speaker, Robert Kowalski, from Imperial College London.

We are also thankful to the organizers of the contest: Jürgen Dix, from the Technical University of Clausthal, and Mehdi Dastani, from the University of Utrecht, to Kostas Stathis for the management of local organisation and for the prompt solution of innumerable logistic problems, and to the technical staff for the web support.

Our thanks also go to the authors who responded to our call with very high quality submissions, and to the members of the CLIMA VI Programme Committee for their valuable work in the limited time available for reviewing and discussing the submitted articles.

Finally, we are greateful to our sponsors: the EU-funded SOCS Project, IST-2001-32530, AgentLink III, and the Association for Logic Programming, for supporting the attendance to CLIMA of 18 among students and young researchers coming from all over the world.

We hope that CLIMA VI will provide an inspiring forum to present state-of-the-art research and further promote Computational Logic in Multi-Agent Systems, and to discuss and confront techniques and approaches to Computational Logic/Multi-Agent Systems problem modelling and solving.

June 2005

Francesca Toni
Paolo Torroni

Workshop Chairs

Francesca Toni, Imperial College London, UK
Paolo Torroni, University of Bologna, Italy

Program Committee

José Alferes, New University of Lisbon, Portugal
Rafael Bordini, University of Durham, UK
Gerd Brewka, University of Leipzig, Germany
Jürgen Dix , Technical University of Clausthal, Germany
Thomas Eiter, Vienna University of Technology, Austria
Klaus Fisher, DFKI, Germany
Michael Fisher, The University of Liverpool, UK
James Harland, Royal Melbourne Institute of Technology, Australia
Katsumi Inoue, National Institute of Informatics, Japan
Antonis Kakas, University of Cyprus
Evelina Lamma, University of Ferrara, Italy
João Leite, New University of Lisbon, Portugal
Paolo Mancarella, University of Pisa, Italy
Paola Mello, University of Bologna, Italy
John-Jules Ch. Meyer, Utrecht University, The Netherlands
Leora Morgenstern, IBM, USA
Wojciech Penczek, Polish Academy of Sciences, Poland
Jeremy Pitt, Imperial College, London, UK
Enrico Pontelli, New Mexico State University, USA
Fariba Sadri, Imperial College London, UK
Ken Satoh, National Institute of Informatics, Japan
Renate Schmidt, The University of Manchester, UK
Tran Cao Son, New Mexico State University, USA
Kostas Stathis, City University London, UK
Wiebe van der Hoek, The University of Liverpool, UK
Cees Witteveen, Delft University of Technology, The Netherlands

CLIMA Steering Committee

Jürgen Dix , Technical University of Clausthal, Germany
Michael Fisher, The University of Liverpool, UK
João Leite, New University of Lisbon, Portugal
Fariba Sadri, Imperial College London, UK
Ken Satoh, National Institute of Informatics, Japan
Francesca Toni, University of Pisa, Italy
Paolo Torroni, University of Bologna, Italy

Contest Organisers

Mehdi Dastani, Utrecht University, The Netherlands
Jürgen Dix, Technical University of Clausthal, Germany

Additional CLIMA Reviewers

Federico Banti
Andrea Bracciali
Andreas Brüning
Lisette van der Burgh
Federico Chesani
Pierangelo Dell'Acqua
Ullrich Hustadt
Magdalena Kacprzak
Sławomir Lasota
Adriaan ter Mors
Yasuo Nagai
Brendan Neville
Ian Pratt-Hartmann
Daniel Ramirez-Cano
Andrzej Szałas
Giacomo Terreni
Satoshi Tojo
Krzysztof Trojanowski
Gregory Weeler
Mathijs de Weerdt

Contest Evaluation Committee

Marco Alberti
Federico Chesani
Mehdi Dastani
Jürgen Dix
Marco Gavanelli
Kostas Stathis
Francesca Toni
Paolo Torroni

Web Support

Fabio Bucciarelli
Federico Chesani

Local Organisation

Kostas Stathis

Sponsoring Institutions

Table of Contents

Combining logic-based agents and decision theory

Robert A. Kowalski

Department of Computing
Imperial College London
180 Queen's Gate
London SW7 2BZ, United Kingdom
rak@doc.ic.ac.uk

Proactive and reactive thinking are different ways of generating candidate actions. Proactive thinking generates actions by reducing goals to sub-goals, and reactive thinking generates actions in response to observed changes in the environment. Both kinds of thinking can be combined in logic-based agent models. Deciding what actions to perform is a separate activity, which might involve the use of logic or not.

Decision theory and game theory, in contrast, focus on the problem of deciding between alternative candidate actions, assuming the alternatives are already given, together with their alternative expected outcomes and their probabilities and utilities. Normally, there is no logic involved.

It is straight-forward, therefore, to combine logic-based agent models and decision theory, by simply using logic to generate candidate actions and using decision theory to decide what actions to perform. However, this does address the problem of generating the expected outcomes of actions and their associated probabilities and utilities. In this talk I will show how logic can be used for some of these purposes. The resulting agent model combines proactive and reactive thinking with a kind of "preactive thinking," which can be characterised as "thinking before you act".

BDI Agent Programming in AgentSpeak using *Jason*

Rafael H. Bordini

Department of Computer Science
University of Durham
Durham DH1 3LE, United Kingdom
`R.Bordini@durham.ac.uk`

The BDI agent architecture has been a central theme in the multi-agent systems literature since the early 90's. After a period of relative declination, it seems BDI agents are back in vogue, with various conference papers referring again to elements of the BDI theory. Arguably, that theory provides the grounding for some of the essential features of autonomous agents and multi-agent systems, so it will always have an important role to play in the research in this area. Besides, the software industry is beginning to use technologies that clearly derived from the academic work on BDI-based systems.

AgentSpeak is one of the most elegant and coherent attempts to define a logic programming language based on the BDI architecture. However, in its original definition, AgentSpeak was not but an abstract programming language. For these reasons, our effort in developing Jason was very much directed towards using AgentSpeak as the basis, but also providing various extensions that are required for the practical development of multi-agent systems.

This tutorial aims at giving an overview of the various features available with Jason. The elegance of the AgentSpeak core of the language interpreted by Jason makes it an interesting tool both for teaching multi-agent systems as well as the practical development of multi-agent systems (in particular in association with existing agent-oriented software engineering methodologies for BDI-like systems). Jason is implemented in Java and is available Open Source at `http://jason.sourceforge.net`.

The $\mathcal{KGP}$ Model of Agency

Fariba Sadri[1] and Kostas Stathis[2]

[1] Department of Computing
Imperial College London
180 Queen's Gate
London SW7 2BZ, United Kingdom
`fs@doc.ic.ac.uk`
[2] Department of Computing
School of Informatics
City University
London EC1V OHB, United Kingdom
`kostas@soi.city.ac.uk`

The $\mathcal{KGP}$ model of agency has been designed with the aim of specifying situated agents that can

- deal with dynamic environments
- adapt to changes in their environment
- dynamically decide their goals
- plan for their goals and interleave action execution with partial planning
- make and record observations from the environment
- react to these observations, where necessary, by re-examining their goals or adjusting their goals and plans
- communicate with other agents.

The model is highly modular and hierarchical. First there is collection of capabilities, including planning, reactivity, goal decision, temporal reasoning and sensing. These are then utilised within transitions that change the state of the agent. The transitions, in turn, are regulated within dynamic context-dependent cycle theories. There is a modular collection of knowledge bases that are used by the capabilities and which record the observations the agent makes.

All the components of the model, including its control theory, are defined using computational logic, some are defined using abductive logic programming, and others, logic programming with priorities. The declarative model is equipped with a provably correct computational model, which is implemented within what is called the PROSOCS platform. The model has been used in applications including combinatorial auctions and negotiation.

In this tutorial we will give an overview and demonstrate some of the main features of the model.

Programming Rational Agent Groups using Executable Logics

Michael Fisher

Department of Computer Science
University of Liverpool
Liverpool L69 3BX, United Kingdom
M.Fisher@csc.liv.ac.uk

Computational power is increasing, and increasingly available, for example via the development of ubiquitous computing. Once large numbers of computational elements can communicate with each other, via wireless networks or the World-Wide Web, then new problems arise in engineering software for such systems. By representing these computational elements as agents, we can provide a simple and intuitive metaphor for both individual computation and that within multi-agent systems. However, software designers need to have appropriate and semantically clear mechanisms for controlling not only how individual agents adapt and evolve, but also how agents interact and combine to form new systems. Without this control not only will the practical development of complex multi-agent systems remain difficult, but agents themselves will not be trusted for use in critical applications.

In this tutorial, I will outline our work on developing an agent programming languages based upon executable temporal logic. In particular, I will describe:

1. the implementation of individual agents through the direct execution of a formal description of individual (rational) agent behaviour given using a combination of temporal logic and logics concerning belief and ability;
2. the core notion of agent groups and how such groups relate to individual agents;
3. the ways in which (ubiquitous/pervasive) multi-agent applications might be developed, by utilising the combination of executable logic (as in 1) and group evolution (as in 2).

This tutorial covers work carried out over a number of years, where we have attempted to use intuitive logical aspects to provide a simple, but effective, mechanism for describing agent computation. It is our assertion that computational logic in general (and executable temporal logics in particular) can provide an appropriate tool for studying not only verifiable agent descriptions, but also novel concepts that can form the basis for the future programming of agent-based systems.

The work presented in this tutorial can be traced back to previous work produced with a number of co-authors on executable temporal logics [1, 2], programming rational agents [3, 4] and programming multi-agent computation [5–8].

References

1. M. Fisher, Representing and Executing Agent-Based Systems. In *Intelligent Agents, ECAI-94 Workshop on Agent Theories, Architectures, and Languages, Amsterdam, The Netherlands, August 8-9, 1994, Proceedings.* Volume 890 of LNAI, Springer-Verlag, 1995.
2. H. Barringer, M. Fisher, D. Gabbay, R. Owens, and M. Reynolds, The Imperative Future: Principles of Executable Temporal Logics. Research Studies Press, 1996.
3. M. Fisher, Implementing BDI-like Systems by Direct Execution. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Morgan-Kaufmann, 1997.
4. M. Fisher and C. Ghidini, Programming Resource-Bounded Deliberative Agents. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI).* Morgan-Kauffman, 1999.
5. M. Fisher and T. Kakoudakis, Flexible Agent Grouping in Executable Temporal Logic. In *Proceedings of the Twelfth International Symposium on Languages for Intensional Programming (ISLIP).* World Scientific Press, March 2000.
6. M. Fisher and C. Ghidini, The ABC of Rational Agent Modelling. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS). Bologna, Italy.* ACM Press, July 2002.
7. M. Fisher, C. Ghidini, and B. Hirsch, Organising Logic-Based Agents. In *Proceedings of the Second Goddard Workshop on Formal Approaches to Agent-Based Systems (FAABS).* Volume 2699 of LNAI, Springer-Verlag, 2003.
8. B. Hirsch, M. Fisher, C. Ghidini, P. Busetta, Organising Software in Active Environments. In *Fifth International Workshop on Computational Logic in Multi-Agent Systems (CLIMA V), Lisbon, Portugal, September 29-30, 2004, Selected, Revised, and Invited Papers.* Volume 3487 of LNAI, Springer-Verlag, 2005.

Specification and Verification of Agent Interaction using $\mathcal{SOCS}$-$\mathcal{SI}$

Federico Chesani[1] and Marco Gavanelli[2]

[1] DEIS, University of Bologna
V.le Risorgimento 2
40136 Bologna, Italy
`fchesani@deis.unibo.it`
[2] Dipartimento di Ingegneria
University of Ferrara
Via Saragat, 1
44100 Ferrara, Italy
`mgavanelli@ing.unife.it`

The definition of the agent interaction space is one of the key aspects of multi-agent systems design. Agent interaction specification has several facets: syntax, semantics, conformance verification and proof of properties. In open societies, heterogenous agents can participate without showing any credentials. Access to their internals or their knowledge bases is typically not permitted, which makes it impossible to know a-priori whether or not agents will behave according to the society interaction rules. On the other hand, when specifying such rules, it is important to know what properties hold in the society when all agents comply with them.

Social Integrity Constraints are the core element of a language proposed within the SOCS project as a means to define interactions in open societies. The proposed language and its abductive interpretation allows the designer to define open, extensible and not over-constrained protocols. Along with the definition language, the $\mathcal{S}$CIFF proof-procedure and a software tool, named *SOCS*-$\mathcal{SI}$, have been developed with the purposes of supporting execution-time verification of agent behaviour with respect to the defined protocols, and proof of protocol properties.

In this tutorial we introduce the theory and the tools (in particular *SOCS*-$\mathcal{SI}$) that can be used to design, define and test interaction protocols in open agent societies. An introduction to *SOCS*-$\mathcal{SI}$ is included in the paper "Compliance Verification of Agent Interaction: a Logic-Based Tool".

Norms and Institutions in Agent Societies: the Language $(\mathcal{C}/\mathcal{C}^+)^{++}$

Marek J. Sergot

Department of Computing
Imperial College London
180 Queen's Gate
London SW7 2BZ, United Kingdom
mjs@doc.ic.ac.uk

The action language $\mathcal{C}^+$ of Giunchiglia, Lee, Lifschitz, McCain, and Turner is a formalism for specifying and reasoning about the effects of actions and the persistence (inertia) of facts over time. An action description in $\mathcal{C}^+$ is a set of $\mathcal{C}^+$ laws which define a labelled transition system of a certain kind.

$(\mathcal{C}^+)^{++}$ is an extended form of $\mathcal{C}^+$ designed for representing norms of behaviour and institutional aspects of (human or computer) societies. There are two main extensions. The first is a means of expressing counts as relations between actions, also referred to as conventional generation. The second extension is a way of specifying the permitted (acceptable, legal) states of a transition system and its permitted (acceptable, legal) transitions. There are implementations supporting a wide range of temporal reasoning and planning tasks (based on the Causal Calculator for $\mathcal{C}^+$ from the University of Texas), for obtaining event calculus like computations with $\mathcal{C}^+$ and $(\mathcal{C}^+)^{++}$ action descriptions, and for using $\mathcal{C}^+$ and $(\mathcal{C}^+)^{++}$ with standard model checking systems for verifying system properties expressed in temporal logics such as CTL.

Programming Multi-Agent Systems in *Qu-Prolog*

Keith Clark and Silvana Zappacosta-Amboldi

Department of Computing
Imperial College London
180 Queen's Gate
London SW7 2BZ, United Kingdom
{clk|sza}@doc.ic.ac.uk

Qu-Prolog is a multi-threaded and distributed version of Prolog with support for high-level communication communication between the threads. Agents can communicate using a thread-to-thread term messages, a publish and subscribe event notification system, or via the the shared Prolog dynamic database. The first two allow threads in different *Qu-Prolog* process any where on the internet to communication, the last one is restricted to threads in the same process.

The tutorial will introduce the three styles of communication and illustrate them using at least two multi-agent applications. *Qu-Prolog* is used at Imperial College as the programming language for practicals in a Multi-agent systems course. The tutorial is based on our experience of its use on the course.

An introduction to some of the features and uses of *Qu-Prolog* is included in the paper at: `http://www.doc.ic.ac.uk/~klc/qp.html`.

Multi-Agent Systems in Logic Programming: Challenges and Outcomes of the SOCS project

Francesca Toni[1] and Paolo Torroni[2]

[1] Department of Computing
Imperial College London
180 Queen's Gate
London SW7 2BZ, United Kingdom
ft@doc.ic.ac.uk
[2] DEIS, University of Bologna
V.le Risorgimento 2
40136 Bologna, Italy
paolo.torroni@unibo.it

The SOCS project is concerned with a computational logic model for the description, analysis and verification of global and open *Societies Of heterogeneous ComputeeS*, where *computees* are computational entities, that is agents realised in computational logic.

SOCS is funded by the European Commission under the Fifth Framework, Future and Emerging Technologies programme, within the Global Computing proactive initiative. The SOCS consortium is composed of six European partners, based in Italy, UK, and Cyprus. These are, respectively, the universities of Pisa, Bologna and Ferrara (Italy), Imperial College London and City University London (UK), and Cyprus University. The project is coordinated by Imperial College London. The project started in January 2002 and will finish in June 2005. Its aims are:

- To provide a computational logic model for the description, analysis and verification of global and open societies of heterogeneous computees, intended as abstractions of the entities that populate open and global computing environments.
- To provide prototype implementations of computees and their societies.
- To run experiments based on various scenaria to ground and test the model.

Details of the project can be found at the project web-site: http://lia.deis.unibo.it/research/socs/

The project aims at using Logic Programming (LP) techniques, such as Abductive Logic Programming (ALP), Constraint Logic Programming (CLP) and Logic Programming with Priorities (LPP), all appropriately extended and integrated to deal with agent problems and Global Computing challenges. We are at the end of the project and we have so far developed an agent model and a society model, both with computational counterparts, together with prototype implementations of both, and we have worked on the formal specification and verification of properties of the models and experimentation with the prototypes. The aim of this SOCS project dissemination event is to give an overview of the project, its main challenges and outcomes.

1 Logic Programming-based Operational Models for Agents and Multi-Agent Systems

We will give an overview of the operational models for individual KGP agents and for societies of agents proposed by the SOCS project. These models are heavily based upon proof procedures for (various extensions of) logic programming. In particular, the operational model for KGP agents relies upon CIFF, a proof procedure for abductive logic programming with constraints, and Gorgias, for logic programming with priorities. The operational model for agent societies instead relies upon SCIFF, a proof procedure for abductive logic programming with arbitrarily quantified variables, CLP constraints, dynamic event handling and reasoning with expectations.

These procedures have been obtained by adapting and suitably extending two existing proof procedures for logic programming, namely Fung and Kowalski's IFF procedure for abductive logic programming, for CIFF and SCIFF, and Kakas and Toni's argumentation-based procedure for negation as failure in logic programming, for Gorgias.

The overall operational models are sound and (in some cases) complete with respect to the abstract KGP model and model for societies of agents, respectively, and form a solid bridge between the models and their implementations within the PROSOCS platform. We discuss the motivations for the new proof procedures, arising from the needs of the KGP agent model and the open societies model of Global Computing, as well as current deficiencies and possible future improvements.

2 Formal Properties of Agents and Agent Systems

A great deal of the project activities has been devoted to formalising and studying properties of agents and agent systems. The SOCS approach to properties is formal, and it aims at exploiting the potential of the declarative programming paradigm for giving a precise specification of properties and for allowing their formal verification. Moreover, the double declarative and operational reading of Computational Logic supports both an abstract description of systems and their (expected) properties, and mechanisms to implement them. Descriptions and mechanisms are closely related to each other so that properties enjoyed by the models are easy to be reflected into implementations.

The kinds of properties investigated span over a relatively broad range. They can roughly be classified into four categories: (1) properties of proof procedures (2) properties of individual computees (3) properties of the social infrastructure, and (4) properties related to protocol conformance. These properties help showing the effectiveness of the computational logic approach in modelling computees and societies, in the sense of facilitating formalisation of properties and prediction of behaviour without the need to resort to empirical methods. They also help exploring the consequences of our design choices.

3 Evaluating Intelligent Systems of Reasoning Agents

The problem of evaluating a project's outcomes is not a trivial one. We have seen how properties can serve to draw some considerations on the effectiveness of the SOCS approach. Indeed, the possibility to specify and verify properties, and a direct relationship between declarative model and implementation, are very important points in favour. However, by pursuing research on computational logic-based multi-agent systems, we situate our work at the intersection of foundational and applied research. The broad and relatively young multi-agent systems community has not yet reached the stage where there are universally accepted benchmark and evaluation criteria, and it is hard to imagine that they will be there in the near future. There exist some attempts in this direction, like robo-cup and the trading agent competition, but they rather aim at addressing a specific problem, and do not define measures for agent autonomy, pro-activeness, reactivity, situatedness, social abilities: all distinguishing aspects of the agent metaphor. On the other hand, computational logic is also a broad research area, but comparatively more focussed in terms of expected results, in that soundness, termination, and sometimes completeness are clearly aims that are common to most proposals in literature.

How do these two perspectives and experiences combine, when it comes to evaluating formal, logic-based research on multi-agent systems? In the context of the SOCS project, born in and motivated by the global computing vision of open societies of heterogeneous agents, beside an evaluation of logic tools (declarative models and proof-procedures) in terms of formal properties, mainly as it concerns specification and verifiability, we considered openness and heterogeneity as two important criteria. We did not define a measure for agent autonomy and other aspects of agency, but we rather tried to define scenarios, where intelligent and social behaviour is required to achieve particular goals. The performance of the PROSOCS implementation has been thoroughly tested through specialised and integrated experiments, in diverse settings such as combinatorial auctions and collaborative problem solving. In this final part of the SOCS dissemination event, we will present the project evaluation setting, and aim to foster a discussion about logic-based multi-agent systems evaluation.

Verification of protocol conformance and agent interoperability[⋆]

Matteo Baldoni, Cristina Baroglio, Alberto Martelli, and Viviana Patti

Dipartimento di Informatica — Università degli Studi di Torino
C.so Svizzera, 185 — I-10149 Torino (Italy)
{baldoni,baroglio,mrt,patti}@di.unito.it

Abstract. In open multi-agent systems agent interaction is usually ruled by public protocols defining the rules the agents should respect in message exchanging. The respect of such rules guarantees interoperability. Given two agents that agree on using a certain protocol for their interaction, a crucial issue (known as "a priori conformance test") is verifying if their interaction policies, i.e. the programs that encode their communicative behavior, will actually produce interactions which are conformant to the agreed protocol. An issue that is not always made clear in the existing proposals for conformance tests is whether the test preserves agents' capability of interacting, besides certifying the legality of their possible conversations. This work proposes an approach to the verification of a priori conformance, of an agent's conversation policy to a protocol, which is based on the theory of formal languages. The conformance test is based on the acceptance of both the policy and the protocol by a special finite state automaton and it guarantees the interoperability of agents that are individually proved conformant. Many protocols used in multi-agent systems can be expressed as finite state automata, so this approach can be applied to a wide variety of cases with the proviso that both the protocol specification and the protocol implementation can be translated into finite state automata. In this sense the approach is general. Easy applicability to the case when a logic-based language is used to implement the policies is shown by means of a concrete example, in which the language DyLOG, based on computational logic, is used.

1 Introduction

Multi-agent systems (MASs) often comprise heterogeneous components, that differ in the way they represent knowledge about the world and about other agents, as well as in the mechanisms used for reasoning about it. Protocols rule the agents' interaction. Therefore, they can be used to check if a given agent can, or cannot, take part into the system. In general, based on this abstraction, open

[⋆] This research is partially supported by MIUR Cofin 2003 "Logic-based development and verification of multi-agent systems" national project and by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779.

systems can be realized, in which new agents can dynamically join the system. The insertion of a new agent in an execution context is determined according to some form of reasoning about its behaviour: it will be added provided that it satisfies the body of the rules within the system, intended as a society.

In a protocol-ruled system of this kind, it is, however, not necessary to check the interoperability (i.e. the capability of *actually* producing a conversation) of the newly entered agent with the other agents in the system if, as long as the rules are satisfied, the property is guaranteed. The problem which amounts to verify if a given *implementation* (an agent interaction policy) respects a given *abstract protocol definition* is known as *conformance* testing. A conformance test can, then, be considered as a tool that, by verifying that agents respect a protocol, *should certify* their interoperability. In this perspective, we expect that two agents which conform to a protocol will *produce a conversation*, that is legal (i.e. correct w.r.t. the protocol), when interacting with one another.

The design and implementation of interaction protocols are crucial steps in the development of a MAS [18, 19]. Following [17], two tests must be executed in the process of interaction protocol engineering. One is the already mentioned conformance test, the other is the *validation* test, which verifies the consistency of an *abstract protocol definition* w.r.t. the *requirements*, derived from the analysis phase, that it should embody. In the literature validation has often been tackled by means of model checking techniques [7, 6, 22], and two kinds of conformance verifications have been studied: *a priori* conformance verification, and *run-time* conformance verification (or compliance) [9, 10, 15]. If we call a *conversation* a specific interaction between two agents, consisting only of communicative acts, the first kind of conformance is a property of the *implementation as a whole* –intuitively it checks if an agent will never produce conversations that violate the abstract interaction protocol specification– while the latter is a property of the *on-going conversation*, aimed at verifying if *that* conversation is legal.

In this work we focus on a priori conformance verification, defining a conformance test, based on the acceptance, of both the agent's policy and the public protocol, by a special finite state automaton. Many protocols used in multi-agent systems can be expressed as finite state automata, so this approach can be applied to a wide variety of cases with the proviso that both the protocol specification and the protocol implementation (policy) can be translated into finite state automata. In this sense the approach is general.

The test that we defined guarantees agent interoperability (see Theorem 1 in Section 3). The communicative behavior of an agent (the decision of which specific action to take) normally relies also on information like the private state of the agent and the social commitments. We will see that our notion of conformance is *orthogonal* to the framework in which one reasons about communication (mentalistic or social [15]). So, our approach works on sets of conversations without caring about the information used to obtain them.

The application of our approach is particularly easy in case a logic-based declarative language is used to implement the policies. In logic languages indeed policies are usually expressed by Prolog-like rules, which can be easily converted in a formal language representation. In Section 4 we show this by means of a concrete example where the language DyLOG [5], based on computational logic, is used for implementing the agents' policies. On the side of the protocol specification languages, currently there is a great interest in using informal, graphical languages (e.g. UML-based) for specifying protocols and in the translation of such languages in formal languages [8, 11]. By this translation it is, in fact, possible to prove properties that the original representation does not allow. In this context, in [4] we have shown an easy algorithm for translating AUML sequence diagrams to finite state automata thus enabling the verification of conformance.

In [4] we already faced the problem of a priori conformance verification as a verification of properties of formal languages, but proposing a different approach with some limitations due to focussing on the legality issue. In fact, interpreting (as we did) the conformance test as the verification that all the conversations, allowed by an agent's policy, are also possible according to the protocol specification, does not entail interoperability. The next section is devoted to explain the expected relations among conformance and the crucial interoperability issue.

2 Conformant and interoperable agents

A *conversation policy* is a program that defines the communicative behavior of a specific agent, implemented in some programming language. A conversation *protocol* specifies the desired communicative behavior of a set of agents and it can be specified by means of many formal tools, such as (but not limited to) Petri nets, AUML sequence diagrams, automata.

In this work we face the problem of conformance verification and interpret a priori conformance as a property that relates two formal languages, the language of the conversations allowed by the conversation policy of an agent, and the language of the conversations allowed by the specification of a communication protocol. They will respectively be denoted by $L(p^{ag}_{lang})$ and $L(p_{spec})$, where *spec* is the specification language, *lang* is the language in which the policy executed by agent *ag* is written, and *p* is the name of the policy or of the protocol at issue. The assumption that we make throughout this paper is that the two languages are *regular sets*. This choice restricts the kinds of protocols to which our proposal can be applied, because finite state automata cannot represent concurrent operations, however, it is still significant because a wide family of protocols (and policies) of practical use can be expressed in a way that can be mapped onto such automata. Moreover, the use of regular sets ensures decidability. Another assumption is that the conversation protocol encompasses only *two agents*. The extension to a greater number of agents will be tackled as future work. Notice that when the MAS is *heterogeneous*, the agents might be implemented in *different languages*.

A conversation protocol specifies the sequences of speech acts that can possibly be exchanged by the involved agents, and that we consider as legal. In agent languages that account for communication, speech acts often have the form $m(ag_s, ag_r, l)$, where m is the performative, ag_s (sender) and ag_r (receiver) are two agents and l is the message content. It is not restrictive to assume that speech acts have this form and to assume that conversations are sequences of speech acts of this form. Depending on the semantics of the speech acts, the conversation will take place in a framework based either on the *mentalistic* or on the *social state* approach [12, 21, 14]. The speech acts semantics, actually, does not play a part in our proposal, which fits both the approaches.

In the following analysis it is important to distinguish the incoming messages, w.r.t. a specific agent ag of the MAS, from the messages uttered by it. We respectively denote the former, where ag plays the role of the receiver, by $\mathsf{m}(\overleftarrow{ag})$, and the latter, where ag is the sender, by and $\mathsf{m}(\overrightarrow{ag})$. We will also simply write $\overleftarrow{\mathsf{m}}$ (*incoming message*) and $\overrightarrow{\mathsf{m}}$ (*outgoing message*) when the agent that receives or utters the message is clear from the context. Notice that these are just short notations, that underline the *role* of a given agent from the *individual perspective* of *that* agent. So, for instance, $m(ag_s, ag_r, l)$ is written as $\mathsf{m}(\overleftarrow{ag_r})$ from the point of view of ag_r, and $\mathsf{m}(\overrightarrow{ag_s})$ from the point of view of the sender but the three notions denote the same object.

A *conversation*, denoted by σ, is a sequence of speech acts that represents a dialogue of a set of agents. We say that a conversation is legal w.r.t. a protocol specification if it respects the specifications given by the protocol. Since $L(p_{spec})$ is the set of all the legal conversations according to p, the definition is as follows.

Definition 1 (Legal conversation). *We say that a conversation σ is legal w.r.t. a protocol specification p_{spec} when $\sigma \in L(p_{spec})$.*

We are now in position to explain, with the help of a few simple examples, the intuition behind the terms "conformance" and "interoperability", that we will, then, formalize.

> *Interoperability is the capability of an agent of actually producing a conversation when interacting with another.*

Often the introduction of a new agent in an execution context is determined according to some form of reasoning about its behaviour: it will be added provided that it satisfies the body of the rules within the system, intended as a society. As long as the rules are satisfied, the property is guaranteed and it will not be necessary to verify interoperability with the single components of the system. This can be done by checking the communicative behavior of the agent against the rules of the society, i.e. against an *interaction protocol.* Such a proof is known as *conformance test.* Intuitively, this test must guarantee the following *definition of interoperability.* This work focuses on it.

> *We expect that two agents, that conform to a protocol, will produce a legal conversation, when interacting with one another.*

Let us begin with considering the following case: suppose that the communicative behavior of the agent ag is defined by a policy that accounts for two conversations $\{\mathsf{m}_1(\overrightarrow{ag})\mathsf{m}_2(\overleftarrow{ag}), \mathsf{m}_1(\overrightarrow{ag})\mathsf{m}_3(\overleftarrow{ag})\}$. This means that after uttering a message m_1, the agent expects one of the two messages m_2 or m_3. Let us also suppose that the protocol specification only allows the first conversation, i.e. that the only possible incoming message is m_2. Is the policy conformant? According to Def. 1 the answer should be no, because the policy allows an illegal conversation. Nevertheless, when the agent will interact with another agent that is conformant to the protocol, the message m_3 will never be received because the other agent will never utter it. So, in this case, we would like the a priori conformance test to accept the policy as *conformant* to the specification.

Talking about incoming messages, let us now consider the symmetric case, in which the *protocol specification* states that after the agent ag has uttered m_1, the other agent can alternatively answer m_2 or m_4 (agent ag's policy, instead, is the same as above). In this case, the expectation is that ag's policy is *not conformant* because, according to the protocol, there is a possible legal conversation (the one with answer m_4) that can be enacted by the *interlocutor* (which is not under the control of ag), which, however, ag cannot handle. So it does not comply to the specifications.

> *As a first observation we expect the policy to be able to handle any incoming message, foreseen by the protocol, and we ignore those cases in which the policy foresees an incoming message that is not supposed to be received at that point of the conversation, according to the protocol specification.*

Let us, now, suppose that agent ag's policy can produce the following conversations $\{\mathsf{m}_1(\overleftarrow{ag})\mathsf{m}_2(\overrightarrow{ag}),\ \mathsf{m}_1(\overleftarrow{ag})\mathsf{m}_3(\overrightarrow{ag})\}$ and that the set of conversations allowed by the protocol specification is $\{\mathsf{m}_1(\overleftarrow{ag})\mathsf{m}_2(\overrightarrow{ag})\}$. Trivially, this policy is *not conformant* to the protocol because ag can send a message (m_3) that cannot be handled by any interlocutor that is conformant to the protocol.

> *The second observation is that we expect a policy to never utter a message that, according to the specification, is not supposed to be uttered at that point of the conversation.*

Instead, in the symmetric case in which the policy contains only the conversation $\{\mathsf{m}_1(\overleftarrow{ag})\mathsf{m}_2(\overrightarrow{ag})\}$ while the protocol states that ag can answer to m_1 alternatively by uttering m_2 or m_3, *conformance holds*. The reason is that at any point of its conversations the agent will always utter legal messages. The restriction of the set of possible alternatives (w.r.t. the protocol) depends on the agent implementor's own criteria. However, the agent must foresee *at least one* of such alternatives otherwise the conversation will be interrupted. Trivially, the case in which the policy contains only the conversation $\{\mathsf{m}_1(\overleftarrow{ag})\}$ is *not conformant.*

> *The third observation is that we expect that a policy always allows the agent to utter one of the messages foreseen by the protocol at every point of the possible conversations. However, it is not necessary that a policy envisions all the possible alternatives.*

To summarize, at every point of a conversation, we expect that a conformant policy never utters speech acts that are not expected, according to the protocol, and we also expect it to be able to handle any message that can possibly be received, once again according to the protocol. However, the policy is not obliged to foresee (at every point of conversation) an outgoing message for every alternative included in the protocol (but it must foresee at least one of them). Incoming and outgoing messages are, therefore, *not handled in the same way.*

These expectations are motivated by the desire to define a minimal set of conditions which guarantee the construction of a conformance test that guarantess the *interoperability* of agents. Let us recall that one of the aims (often implicit) of conformance is, indeed, interoperability, although sometimes research on this topic restricts its focus to the legality issues. We claim –and we will show– that two agents that respect this minimal set of conditions (w.r.t. an agreed protocol) will *actually* be able to interact, respecting at the same time the protocol. The relevant point is that this certification is a property that can be checked on the single agents, rather than on the agent society. This is interesting in application domains (e.g. web services) with a highly dynamic nature, in which agents are searched for and composed at the moment in which specific needs arise.

3 Conformance test

In order to decide if a policy is conformant to a protocol specification, it is not sufficient to perform an inclusion test; instead, as we have intuitively shown by means of the above examples, it is necessary to prove mutual properties of both $L(p^{ag}_{lang})$ and $L(p_{spec})$. The method that we propose, for proving such properties, consists in verifying that both languages are recognized by a special finite state automaton, whose construction we are now going to explain. Such an automaton is based on the automaton that accepts the *intersection* of the two languages. All the conversations that belong to the intersection are certainly legal. This, however, is not sufficient, because there are further conditions to consider, for instance there are conversations that we mean to allow but that do not belong to the intersection. In other words, the "intersection automaton" does not capture all the expectations reported in Section 2. We will extend this automaton in such a way that it will accept the converations in which the agent expects messages that are not foreseen by the specification as well as those which include outgoing messages that are not envisioned by the policy. On the other hand, the automaton will not accept conversations that include incoming messages that are not expected by the policy nor it will accept conversations that include outgoing messages, that are not envisioned by the protocol (see Fig. 1).

3.1 The automaton M_{conf}

If $L(p^{ag}_{lang})$ and $L(p_{spec})$ are regular, they are accepted by two (deterministic) *finite automata*, that we respectively denote by $M(p^{ag}_{lang})$ and $M(p_{spec})$, that we can assume as having the *same alphabet* (see [16]). An automaton is a five-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite input alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and δ is a transition function mapping $Q \times \Sigma$ to Q. In a finite automaton we can always classify states in two categories: *alive states*, that lie on a path from the initial state to a final state, and *dead states*, the other ones. Intuitively, alive states accept the language of the prefixes of the strings accepted by the automaton.

For reasons that will be made clear shortly, we request the two automata to show the following property: the edges that lead to the same state must *all* be labelled either by incoming messages or by outgoing messages w.r.t. *ag*.

Definition 2 (IO-automaton). *Given an automaton $M = (Q, \Sigma, \delta, q_0, F)$, let $E_q = \{m \mid \delta(p, m) = q\}$ for $q \in Q$. We say that M is an IO-automaton iff for every $q \in Q$, E_q alternatively consists only of incoming or only of outgoing messages w.r.t. an agent ag.*

Notice that an automaton that does not show this property can always be transformed so as to satisfy it, in linear time w.r.t. the number of states, by splitting those states that do not satisfy the property. We will denote a state q that is reached only by incoming messages by the notation $\overleftarrow{q}$ (we will call it an I-state), and a state q that is reached only by outgoing messages by $\overrightarrow{q}$ (an O-state).

Finally, let us denote by $M^{\times}(p^{ag}_{lang}, p_{spec})$ the deterministic finite automaton that accepts the language $L(p^{ag}_{lang}) \cap L(p_{spec})$. It is defined as follows. Let $M(p^{ag}_{lang})$ be the automaton $(Q^P, \Sigma, \delta^P, q^P_0, F^P)$ and $M(p_{spec})$ the automaton $(Q^S, \Sigma, \delta^S, q^S_0, F^S)$:

$$M^{\times}(p^{ag}_{lang}, p_{spec}) = (Q^P \times Q^S, \Sigma, \delta, [q^P_0, q^S_0], F^P \times F^S)$$

where for all q^P in Q^P, q^S in Q^S, and m in Σ, $\delta([q^P, q^S], m) = [\delta^P(q^P, m), \delta^S(q^S, m)]$. We will briefly denote this automaton by $M^{\times}$.

Notice that all the *conversations* that are accepted by $M^{\times}$ are surely *conformant* (Def. 1). For the so built automaton, it is easy to prove the following property.

Proposition 1. *$M^{\times}(p^{ag}_{lang}, p_{spec})$ is an IO-automaton if $M(p^{ag}_{lang})$ and $M(p_{spec})$ are two IO-automata.*

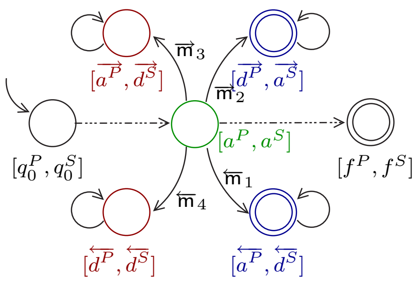
Definition 3 (Automaton M_{conf}). *The finite state automaton $M_{conf}(p^{ag}_{lang}, p_{spec})$ is built by applying the following steps to $M^{\times}(p^{ag}_{lang}, p_{spec})$ until none is applicable:*

(a) if $\overleftarrow{q} = [\overleftarrow{a^P}, \overleftarrow{d^S}]$ in Q is an I-state, such that $\overleftarrow{a^P}$ is an alive state and $\overleftarrow{d^S}$ is a dead state, we set $\delta(\overleftarrow{q}, m) = \overleftarrow{q}$ for every m in Σ, and we put $\overleftarrow{q}$ in F;

(b) if $\overleftarrow{q} = [\overleftarrow{d^P}, \overleftarrow{a^S}]$ in Q is an I-state, such that $\overleftarrow{d^P}$ is dead and $\overleftarrow{a^S}$ is alive, we set $\delta(\overleftarrow{q}, m) = \overleftarrow{q}$ for every m in Σ, without modifying F;

(c) if $\overrightarrow{q} = [\overrightarrow{a^P}, \overrightarrow{d^S}]$ in Q is an O-state, such that $\overrightarrow{a^P}$ is alive and $\overrightarrow{d^S}$ is dead, we set $\delta(\overrightarrow{q}, m) = \overrightarrow{q}$ for every m in Σ (without modifying F);

(d) if $\overrightarrow{q} = [\overrightarrow{d^P}, \overrightarrow{d^S}]$ in Q is an O-state, such that $\overrightarrow{d^P}$ is dead and $\overrightarrow{d^S}$ is alive, we set $\delta(\overrightarrow{q}, m) = \overrightarrow{q}$ for every m in Σ, and we put $\overrightarrow{q}$ in F.



$\overrightarrow{\mathsf{m}}_3$
$[\overrightarrow{a^P}, \overrightarrow{d^S}]$
$[\overrightarrow{d^P}, \overrightarrow{a^S}]$
$\overrightarrow{\mathsf{m}}_2$
$[q_0^P, q_0^S]$
$[a^P, a^S]$
$[f^P, f^S]$
$\overleftarrow{\mathsf{m}}_1$
$\overleftarrow{\mathsf{m}}_4$
$[\overleftarrow{d^P}, \overleftarrow{d^S}]$
$[\overleftarrow{a^P}, \overleftarrow{d^S}]$

Fig. 1. A general schema of the M_{conf} automaton. From bottom-right, anticlockwise, cases (a), (b), (c), and (d).

These four transformation rules can, intuitively, be explained as follows. Rule (a) handles the case in which, at a certain point of the conversation, according to the policy it is possible to receive a message that, instead, cannot be received according to the specification (it is the case of message $\overleftarrow{\mathsf{m}}_1$ in Fig. 1). Actually, if the agent will interact with another agent that respects the protocol, this message can never be received, so we can ignore the paths generated by the policy from the message at issue onwards. Since this case does not compromise conformance, we want our automaton to accept all these strings. For this reason we set the state as final. Rule (b) handles the symmetric case (Fig. 1, message $\overleftarrow{\mathsf{m}}_4$), in which at a certain point of the conversation it is possible, according to the specification, to receive a message, that is not accounted for by the implementation. In this case the state at issue is turned into a trap state (a state that is not final and that has no transition to a different state); by doing so, all the conversations that are foreseen by the specification from that point onwards will not be accepted by M_{conf}. Rule (c) handles the cases in which a message can possibly be uttered by the agent, according to the policy, but it is not possible according to the specification (Fig. 1, message $\overrightarrow{\mathsf{m}}_3$). In this case, the policy is not conformant, so we transform the current state in a trap state. By doing so, part of the conversations possibly generated by the policy will not be accepted by the automaton. The symmetric case (d) (Fig. 1, message $\overrightarrow{\mathsf{m}}_2$), instead, does not prevent conformance, in fact, an agent is free not to utter a message foreseen by the protocol. However, the conversations that can be generated from that point according to the specification are to be accepted as well. For this reason the state is turned into an accepting looping state. Finally, since a policy is expected to envision at least one of the messages forseen by the protocol, we require that for those states $q^S \in Q^S$, that emit edges labelled with outgoing messages, w.r.t. ag, which are part of strings accepted by $M(p_{spec})$ (legal conversations according to the protocol specification), there is at least one $\mathsf{m}(\overrightarrow{ag})$ such that $\delta^S(q^S, \mathsf{m}(\overrightarrow{ag})) = p^S$ and p^S is an *alive state*. We will denote by $Mess_{q^S}$ the set of all such messages. In this case, we say the automaton is *complete*.

Definition 4 (Complete automaton). *We say that the automaton M_{conf} is* complete *iff for all states of form $[q^P, q^S]$ of M_{conf}, such that $Mess_{q^S} \neq \emptyset$, there is a message $\mathsf{m}(\overrightarrow{ag})' \in Mess_{q^S}$ such that $\delta([q^P, q^S], \mathsf{m}(\overrightarrow{ag})')$ is a state of M_{conf} composed of two alive states.*

One may wonder if the application of rules (b) and (c) could prevent the *reachability of states*, that have been set as accepting states by the other two rules. Notice that their application, cannot prevent the reachability of *alive-alive* accepting states, i.e. those that accept the strings belonging to the intersection of the two languages, because all the four rules only work on dead states. If a state has been set as a trap state (either by rule (b) or (c)), whatever conversation is possibly generated after it by the policy is illegal w.r.t. the specification. So it is correct that the automaton is modified in such a way that the policy language is not accepted by it and that the final state cannot be reached any more.

3.2 Conformance and interoperability

We can now discuss how to check that an agent conforms to a given protocol. The following is our definition of conformance test. It guarantees the expectations that we have explained by examples in Section 2.

Definition 5 (Policy conformance test). *A policy p^{ag}_{lang} is conformant to a protocol specification p_{spec} iff the automaton $M_{conf}(p^{ag}_{lang}, p_{spec})$ is complete and it accepts both languages $L(p^{ag}_{lang})$ and $L(p_{spec})$.*

We are now in position to state that a policy that passes the above test can carry on *any* conformant conversation it is involved in.

Proposition 2. *Given a policy p^{ag}_{lang} that is conformant to a protocol specification p_{spec}, according to Def. 5, for every prefix σ' that is common to the two languages $L(p_{spec})$ and $L(p^{ag}_{lang})$, there is a conversation $\sigma = \sigma'\sigma''$ such that σ is in the intersection of $L(p^{ag}_{lang})$ and $L(p_{spec})$.*

Proof. (*sketch*) If σ' is a common prefix, then it leads to a state of the automaton M_{conf} of the kind $[a^P, a^S]$ (see Figure 1). *If* there is a conversation $\sigma = \sigma'\sigma''$ in $L(p^{ag}_{lang})$, then this must be a legal conversation. In fact, let us consider the general schema of M_{conf} in Figure 1. If p^{ag}_{lang} is conformant, $L(p^{ag}_{lang})$ is accepted by M_{conf}. Then, by construction M_{conf} does not contain any state $[\overrightarrow{a^P}, \overrightarrow{d^S}]$ due to illegal messages uttered by the agent nor it contains any state $[\overleftarrow{d^P}, \overleftarrow{a^S}]$ due to incoming messages that are not accounted for by the policy. Obviously, no conversation σ in $L(p^{ag}_{lang})$ can be accepted by states of the kind $[\overrightarrow{d^P}, \overrightarrow{a^S}]$ because the agent does not utter the messages required to reach such states. Finally, no conversation produced by the agent will be accepted by states of the kind $[\overleftarrow{a^P}, \overleftarrow{d^S}]$ if the interlocutor is also conformant to the protocol, because the latter cannot utter illegal messages. Now, at every step after the state $[a^P, a^S]$ mentioned above, due to *policy conformance* all the incoming messages (w.r.t.

the agent) must be foreseen by the policy. Moreover, due to the *completeness* of M_{conf}, in the case of outgoing messages, the policy must foresee at least one of them. Therefore, from $[a^P, a^S]$ it is possible to perform one more common step. **q.e.d.**

Notice that the *intersection* of $L(p^{ag}_{lang})$ and $L(p_{spec})$ cannot be empty because of policy conformance, and also that Proposition 2 does not entail that the two languages coincide (i.e. the policy is not necessarily a full implementation of the protocol). As a consequence, given that the conversation policies of two agents ag_1 and ag_2, playing the different roles of an interaction protocol p_{spec}, are *conformant* to the specification according to Def. 5, and denoting by I the intersection $\cap^{i=1,2}_{ag_i} L(p^{ag_i}_{lang_i})$, we can prove ag_1 and ag_2 *interoperability.* The demonstration is similar to the previous one. Roughly, it is immediate to prove that every prefix, that is common to the two policies, also belongs to the protocol, then, by performing reasoning steps that are analogous to the previous demonstration, it is possible to prove that a common legal conversation must exist when both policies satisfy the conformance test given by Def. 5.

Theorem 1 (Interoperability). *For every prefix σ' that is common to the two languages $L(p^{ag_1}_{lang_1})$ and $L(p^{ag_2}_{lang_2})$, there is a conversation $\sigma = \sigma'\sigma''$ such that $\sigma \in I$.*

Starting from regular languages, all the steps that we have described that lead to the construction of M_{conf} and allow the verification of policy conformance, are decidable and the following theorem holds.

Theorem 2. *Policy conformance is decidable when $L(p^{ag}_{lang})$ and $L(p_{spec})$ are regular languages.*

Notice that if we do not require M_{conf} to be complete, we could not guarantee the *third expectation* reported in Section 2, which requires that, at every state of the conversation, if a role is supposed to utter a message out of a set of possibilities, the agent's policy envisions *at least one* of them. Thus, we could not guarantee that the two agents, playing the two roles of a same protocol, would be able to lead to an end their conversations. On the other hand, the definition would be sufficient to satisfy the first two expectations. In other words, we can prove that whatever conversation is in the intersection I, it is legal, but we cannot prove that I is not empty.

Proposition 3. *All the conversations that a policy p^{ag}_{lang}, that is conformant according to Def. 5 (without requiring M_{conf} to be complete) to a protocol specification p_{spec}, will produce when it interacts with any agent that is equally conformant to p_{spec}, are always legal w.r.t. this protocol, according to Def. 1.*

4 The DyLOG language: a case study

In this section we show how the presented approach particularly fits logic languages, using as a case of study the DyLOG language [5], previously developed in our group. The choice is due to the fact that this language explicitly supplies the tools for representing communication protocols and that we have already presented an algorithm for turning a DyLOG program in a regular grammar (therefore, into a finite state automaton) [4]. This is, however, just an example. The same approach could be applied to other logic languages. For this reason we will confine its description to the strict necessary.

DyLOG [5] is a logic programming language for modeling rational agents, based upon a modal logic of actions and mental attitudes, in which modalities represent actions as well as beliefs that are in the agent's mental state. It accounts both for atomic and complex actions, or procedures, for specifying the agent behavior. DyLOG agents can be provided with a *communication kit* that specifies their communicative behavior [3]. In DyLOG *conversation policies* are represented as procedures that compose speech acts (described in terms of their preconditions and effects on the beliefs in the agent's mental state). They specify the agent communicative behavior and are expressed as prolog-like procedures of form:

$$p_0 \textbf{ is } p_1; p_2; \ldots; p_m$$

where p_0 is a procedure name, the p_i's in the body are procedure names, atomic actions, or test actions, and ';' is the sequencing operator.

Besides speech acts, protocols can also contain *get message actions*, used to read incoming communications. From the perspective of an agent, expecting a message corresponds to a query for an external input, thus it is natural to interpret this kind of actions as a special case of sensing actions. As such, their outcome, though belonging to a predefined set of alternatives, cannot be predicted before the execution. A get_message action is defined as:

$$\begin{aligned}&\mathsf{get\_message}(ag_i, ag_j, l) \textbf{ is}\\&\qquad \mathsf{speech\_act}_1(ag_j, ag_i, l) \textbf{ or } \ldots \textbf{or } \mathsf{speech\_act}_k(ag_j, ag_i, l)\end{aligned}$$

On the right hand side the finite set of alternative incoming messages that the agent ag_i expects from the agent ag_j in the context of a given conversation. The information that is actually received is obtained by looking at the effects that occurred on ag_i's mental state.

From the specifications of the interaction protocols and of the relevant speech acts contained in the domain description, it is possible to trigger a *planning* activity by executing *existential queries* of form Fs **after** $p_1; p_2; \ldots; p_m$, that intuitively amounts to determine if there is a possible execution of the enumerated actions after which the condition Fs holds. If the answer is positive, a conditional plan is returned. Queries of this kind can be given an answer by a goal-directed proof procedure that is described in [3].

The example that we consider involves a reactive agent. The program of its interlocutor is not given: we will suppose that it adheres to the public protocol specification against which we will check our agent's conformance. The example rephrases one taken from the literature, that has been used in other proposals (e.g. [13]) and, thus, allows a better comprehension as well as comparison. We just set the example in a realistic context. The agent is a web service [2] that answers queries about the movies that are played. Its interlocutor is the requester

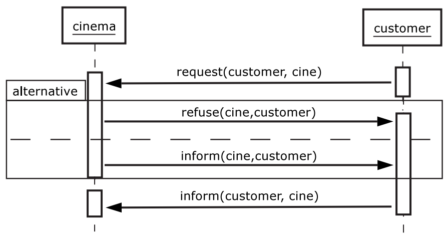


Fig. 2. AUML sequence diagram.

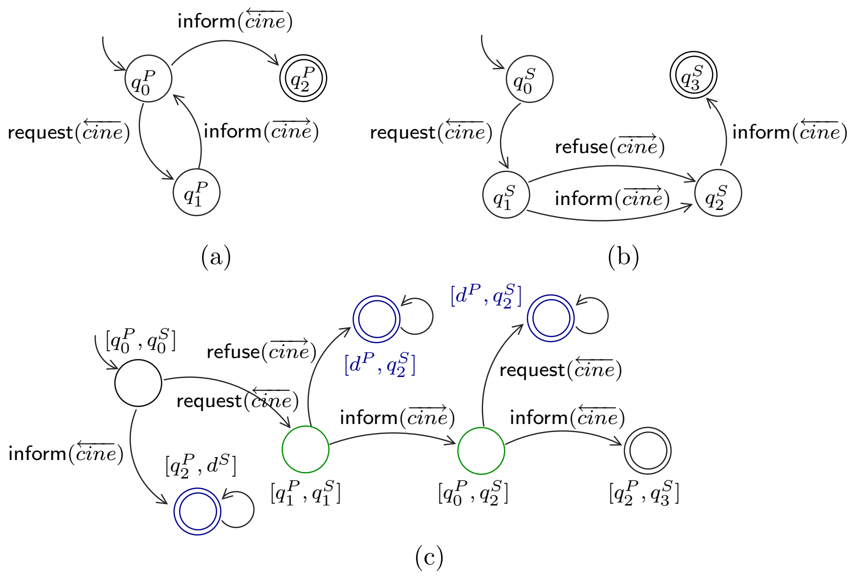
of information (that we do not detail supposing that it respects the agreed protocol). This protocol is described in Fig. 2 as an AUML sequence diagram [20]. It is very simple: the agent that plays the role "cinema" waits for a request from another agent (if a certain movie is played), then, it can alternatively send the requested information (yes or no) or refuse to supply information; the protocol is ended by an acknowledgement message from the customer to the cinema. Hereafter, we consider the implementation of the web service of a specific cinema, written as a DyLOG communication policy. This program has a different aim: it allows answering to a sequence of information requests from the same customer and it never refuses an answer.

(a) $\mathsf{get\_info\_movie}(cine, customer)$ **is**
  $\mathsf{get\_request}(cine, customer, available(Movie))$;
  $\mathsf{send\_answer}(cine, customer, available(Movie))$;
  $\mathsf{get\_info\_movie}(cine, customer)$
(b) $\mathsf{get\_info\_movie}(cine, customer)$ **is** $\mathsf{get\_ack}(cine, customer)$

(c) $\mathsf{send\_answer}(cine, customer, available(Movie))$ **is**
  $\mathcal{B}^{cinema} available(Movie)?$; $\mathsf{inform}(cine, customer, available(Movie))$
(d) $\mathsf{send\_answer}(cine, customer, available(Movie))$ **is**
  $\neg\mathcal{B}^{cinema} available(Movie)?$; $\mathsf{inform}(cine, customer, \neg available(Movie))$

(e) $\mathsf{get\_request}(cine, customer, available(Movie))$ **is**
  $\mathsf{request}(customer, cine, available(Movie)$
(f) $\mathsf{get\_ack}(cine, customer, ack)$ **is** $\mathsf{inform}(customer, cine, ack)$

The question that we try to answer is whether this policy is *conformant* to the given protocol, and we will discuss whether another agent that plays as a customer and that is proved conformant to the protocol will actually be able to *interoperate* with this implementation of the cinema service. For what concerns the AUML sequence diagram, we have proved in [4] that diagrams containing only message, alternative, loop, exit, and reference to a subprotocol operators can be represented as a right-linear grammar, that generates a regular language. The automaton reported in Fig. 3(b) is obtained straightforwardly from this grammar. For what concerns the implementation, by applying the results reported in [4] it is possible to turn a DyLOG program in a context-free language. This grammar captures the structure of the possible conversations disregarding the semantics of the speech acts. When we have only right-recursion in the program, then, the obtained grammar is right-linear. So also in this case a regular language is obtained, hence the automaton in Fig. 3(a). Notice that all the three automata are represented from the perspective of agent *cine*, so all the short notation for the messages are to be interpreted as incoming or outgoing messages w.r.t. this agent.



inform($\overleftarrow{cine}$)
q_0^P
q_2^P
request($\overleftarrow{cine}$)
inform($\overrightarrow{cine}$)
q_1^P
(a)
q_0^S
q_3^S
request($\overleftarrow{cine}$)
refuse($\overrightarrow{cine}$)
inform($\overleftarrow{cine}$)
q_1^S
inform($\overrightarrow{cine}$)
q_2^S
(b)
$[q_0^P, q_0^S]$
refuse($\overrightarrow{cine}$)
$[d^P, q_2^S]$
$[d^P, q_2^S]$
request($\overleftarrow{cine}$)
request($\overleftarrow{cine}$)
inform($\overrightarrow{cine}$)
inform($\overleftarrow{cine}$)
inform($\overleftarrow{cine}$)
$[q_2^P, d^S]$
$[q_1^P, q_1^S]$
$[q_0^P, q_2^S]$
$[q_2^P, q_3^S]$
(c)

Fig. 3. (a) Policy of agent *cine*; (b) protocol specification; (c) M_{conf} automaton. Only the part relevant to the discussion is shown.

The protocol allows only two conversations between *cine* and *customer* (the content of the message is not relevant in this example, so we skip it): request(*customer*, *cine*) inform(*cine*, *customer*) inform(*customer*, *cine*) and request(*customer*, *cine*) refuse(*cine*, *customer*) inform(*customer*, *cine*). Let us denote this protocol by get_info_movie$_{AUML}$ (AUML is the specification language).

Let us now consider an agent (*cine*), that is supposed to play as cinena. This agent's policy is described by the above DyLOG program. The agent has a *reactive behavior*, that depends on the message that it receives, and its policy allows an infinite number of conversations of any length. Let us denote this language by get_info_movie$^{cine}_{DyLOG}$. In general, it allows all the conversations that begin with a (possibly empty) series of exchanges of kind request($\overleftarrow{cine}$) followed by inform($\overrightarrow{cine}$), concluded by a message of kind inform($\overleftarrow{cine}$).

To verify its conformance to the protocol, and then state its interoperability with other agents that respect such protocol, we need to build the M_{conf}

automaton for the policy of *cine* and the protocol specification. For brevity, we skip its construction steps and directly report M_{conf} in Fig. 3(c).

Let us now analyse M_{conf} for answering our queries. Trivially, the automaton is complete and it accepts both languages (of the policy and of the specification), therefore, $\mathsf{get\_info\_movie}^{cine}_{DyLOG}$ is policy conformant to $\mathsf{get\_info\_movie}_{AUML}$. Moreover, when the agent interacts with another agent *customer* whose policy is conformant to $\mathsf{get\_info\_movie}_{AUML}$, the messages $\mathsf{request}(\overleftarrow{cine})$ and $\mathsf{inform}(\overleftarrow{cine})$ will not be received by *cine* in all the possible states it expects them. The reason is simple: for receiving them it is necessary that the interlocutor utters them, but by definition (it is conformant) it will not. The fact that $\mathsf{refuse}(\overrightarrow{cine})$ is never uttered by *cine* does not compromise conformance.

5 Conclusions and related work

In this work we propose an approach to the verification of the conformance of an agent's conversation policy to a public conversation protocol, which is based on the theory of *formal languages*. Differently than works like [1], where the compliance of the agents' communicative behavior to the protocol is verified at run-time, we tackled the verification of *a priori* conformance, a property of the *policy* as a whole and not of the on-going conversation only.

This problem has been studied by other researchers, the most relevant analysis probably being the one by Endriss et al. and reported in [10]. Here, the problem was faced in a logic framework; the authors introduce three degrees of conformance, namely *weak*, *exhaustive*, and *robust conformance*. An agent is weakly conformant to a protocol iff it never utters any dialogue move which is not a legal continuation (w.r.t. the protocol) of any state of the dialogue the agent might be in. It is *exhaustively conformant* to a protocol iff it is weakly conformant to it and, for every received legal input, it will utter one of the expected dialogue moves. It is *robustly conformant* iff it is exhaustively conformant and for any illegal input move received it will utter a special dialogue move (such as not-understood) indicating this violation. Under the assumption that in their conversations the agents strictly alternate in uttering messages (ag_1 tells something to ag_2 which answers to ag_1 and so on), Endriss and collegues show that by their approach it is possible to prove *weak conformance* in the case of logic-based agents and shallow protocols [1].

Our *Policy conformance* (Def. 5) guarantees that an agent, at any point of its conversations, can only utter messages which are legal w.r.t. the protocol, because of the M_{conf} construction step, given by rule (c). In this respect it entails *weak conformance* [10], however, our notion of conformance differs from it because it also guarantees that whatever incoming message the agent may receive, in any conversation context, its policy will be able to handle it.

The second very important property that is guaranteed by our proposal is that, given two policies *each* of which is *conformant* to a protocol specification,

[1] A protocol is shallow when the current state is sufficient to decide the next action to perform. This is not a restriction.

their *interoperability* is guaranteed. In other words, we captured the expectation that conformance, a property of the single policy w.r.t. the public protocol, should in some way guarantee agents (legal) interoperability. Interoperability is not mentioned by Endriss et al., who do not formally prove that it is entailed by (all or some of) their three definitions.

Moreover, we do not limit in any way the structure of the conversations (in particular, we do not require a strict alternation of the uttering agents) nor we focus on a specific class of implementation or specification languages. One further characteristic is that, according to Def. 5, a policy may also expect incoming messages, that are not expected by the protocol specification, for this does not prevent the correct interaction with another conformant agent, and it is not requested to implement a whole set of alternative outgoing messages that are considered possible by the protocol.

This work is, actually, a deep revision of the work that the authors presented at [4], where the verification of a priori conformance was faced only in the specific case in which DyLOG [5] is used as the policy implementation language and AUML [20] is used as the protocol specification language. Basically, in that work the idea was to turn the problem into a problem of *formal language inclusion*. The two considered languages are the set of all the possible conversations foreseen by the protocol specification, let us denote it by $L(p_{AUML})$, and the set of all the possible conversations according to the policy of agent ag, let us denote it by $L(p^{ag}_{dylog})$. The conformance property could then be expressed as the following inclusion: $L(p^{ag}_{dylog}) \subseteq L(p_{AUML})$. The current proposal is more general than the one in [4], being independent from the implementation and specification languages. Moreover, as we have explained in the introduction, the interpretation of conformance as an inclusion test is too restrictive, on a hand, and not sufficient to express all the desiderata connected to this term, which are, instead, well-captured by our definitions of policy conformance.

Finally, to the best of our knowledge, in those works that address the problem of verifying the conformance in systems of communicating agents by using model checking techniques (e.g. [13]), the issue of interoperability is not tackled or, at least, this does not clearly emerge. For instance, Giordano, Martelli and Schwind [13] based their approach on the use of a dynamic linear time logic. Protocols are specified, according to a social approach, by means of temporal constraints representing permissions and commitments. Following [15] the paper shows how to prove that an agent is compliant with a protocol, given the program executed by the agent, by assuming that all other agents participating in the conversation are compliant with the protocol, i.e. they respect their permissions and commitments. However, this approach does not guarantee interoperability.

References

1. M. Alberti, D. Daolio, P. Torroni, M. Gavanelli, E. Lamma, and P. Mello. Specification and verification of agent interaction protocols in a logic-based system. In *ACM SAC 2004*, pages 72–78. ACM, 2004.

2. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services*. Springer, 2004.
3. M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. Reasoning about self and others: communicating agents in a modal action logic. In *ICTCS'2003*, volume 2841 of *LNCS*, pages 228–241. Springer, October 2003.
4. M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella. Verifying protocol conformance for logic-based communicating agents. In *Proc. of 5th Int. Workshop on Computational Logic in Multi-Agent Systems, CLIMA V*, LNCS, 2005.
5. M. Baldoni, L. Giordano, A. Martelli, and V. Patti. Programming Rational Agents in a Modal Action Logic. *Annals of Mathematics and Artificial Intelligence, Special issue on Logic-Based Agent Implementation*, 41(2-4):207–257, 2004.
6. J. Bentahar, B. Moulin, J. J. Ch. Meyer, and B. Chaib-Draa. A computational model for conversation policies for agent communication. In *Pre-Proc. of CLIMA V*, pages 66–81, 2004.
7. R. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model Checking AgentSpeak. In *Proc. of AAMAS 2003*, 2003.
8. L. Cabac and D. Moldt. Formal semantics for auml agent interaction protocol diagrams. In *Proc. of AOSE 2004*, pages 47–61, 2004.
9. U. Endriss, N. Maudet, F. Sadri, and F. Toni. Protocol conformance for logic-based agents. In *Proc. of IJCAI-2003*, pages 679–684, 2003.
10. U. Endriss, N. Maudet, F. Sadri, and F. Toni. Logic-based agent communication protocols. In *Advances in agent communication languages*, volume 2922 of *LNAI*, pages 91–107. Springer-Verlag, 2004. invited contribution.
11. R. Eshuis and R. Wieringa. Tool support for verifying UML activity diagrams. *IEEE Trans. on Software Eng.*, 7(30), 2004.
12. FIPA. Fipa 97, specification part 2: Agent communication language. Technical report, FIPA (Foundation for Intelligent Physical Agents), November 1997.
13. L. Giordano, A. Martelli, and C. Schwind. Verifying communicating agents by model checking in a temporal action logic. In *JELIA'04*, volume 3229 of *LNAI*, pages 57–69, Lisbon, Portugal, 2004. Springer-Verlag.
14. F. Guerin. *Specifying Agent Communication Languages*. PhD thesis, Imperial College, London, April 2002.
15. F. Guerin and J. Pitt. Verification and Compliance Testing. In H.P. Huget, editor, *Communication in Multiagent Systems*, volume 2650 of *LNAI*, pages 98–112. Springer, 2003.
16. J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley Publishing Company, 1979.
17. M. P. Huget and J.L. Koning. Interaction Protocol Engineering. In *Communication in Multiagent Systems*, *LNAI* 2650, pages 179–193. Springer, 2003.
18. A. Mamdani and J. Pitt. Communication protocols in multi-agent systems: A development method and reference architecture. In *Issues in Agent Communication*, volume 1916 of *LNCS*, pages 160–177. Springer, 2000.
19. N. Maudet and B. Chaib-draa. Commitment-based and dialogue-based protocols: new trends in agent communication languages. *Knowledge engineering review*, 17(2), 2002.
20. J. Odell, H. V. D. Parunak, and B. Bauer. Extending UML for agents. In *Proc. of the Agent-Oriented Information System Workshop at AAAI'00*. 2000.
21. M. P. Singh. A social semantics for agent communication languages. In *Proc. of IJCAI-98 Workshop on Agent Communication Languages*, Berlin, 2000. Springer.
22. C. Walton. Model checking agent dialogues. In *Int. Workshop on Declarative Agent Languages and Technology*, pages 156–171, 2004.

A Semantics for Agent Communication Languages based on commitments and penalties

Leila Amgoud Florence Dupin de Saint-Cyr

Institut de Recherche en Informatique de Toulouse (I.R.I.T.)–C.N.R.S.
Université Paul Sabatier, 118 route de Narbonne,
31062 Toulouse Cedex 4, France
{amgoud, bannay}@irit.fr

Abstract. In complex multi agent systems, the agents may be heterogeneous and possibly designed by different programmers. Thus, the importance of defining a standard framework for agent communication languages (ACL) with a clear *semantics* has been widely recognized. The semantics should meet two important objectives: *verifiability* and *flexibility*. Classical proposals (*mentalistic semantics* and *social semantics*) fail to meet these objectives.
In this paper we propose a semantics for ACL which is both verifiable and flexible. That semantics is social in nature since it is based on commitments. Two kinds of commitments can be distinguished: i) the commitments made by the agent itself during the dialogue such as promises, and ii) the commitments made by the others to the agent such as requests or questions. To these commitments, we associate two kinds of penalties for sanctioning the agents which do not respect its commitments. The first kind of penalty ensures that the agent is honest whereas the second kind of penalty ensures that the agent is cooperative. These penalties make the semantics verifiable.
Moreover, our semantics is flexible since it is not given on the basis of particular speech acts, but on the well-known *categories* of speech acts identified by Searle in [11, 12]. This makes it more general than the existing ones.

Key words: Agent Communication Languages, Commitment, Penalties.

1 Introduction

When building multi-agent systems, we take for granted the fact that the agents which make up the system will need to communicate and to engage in the different types of dialogue identified by Walton and Krabbe in [15], using a communication language (ACL).
The definition of an ACL from the syntactic point of view (the different *speech acts*[1] that agents can perform during a dialogue) poses no problems. The situation is different when semantics is taken into account. Any communication

[1] The speech acts are also called elsewhere *illocutionary* acts or *performatives.*

language must have a well-defined semantics. Given that agents in a multi-agent system may be independently designed by different programmers, a clear understanding of semantics is essential. Moreover, the semantics should meet two important *properties*: *verifiability* and *flexibility*.
Although a number of agent communication languages have been developed, obtaining a suitable formal semantics for ACLs which satisfies the above objectives remains one of the greatest challenges of multi-agent theory.
There are mainly two categories of semantics: the *mentalistics* semantics and the *social* ones. The mentalistic semantics, used in KQML [7] and FIPA [8], is based on a notion of speech act close to the concept of illocutionary act as developed in speech act theory [4, 12]. The basic idea behind this semantics is to define the conditions under which a given speech act can be played. Unfortunately, the conditions are based on the mental states of the agents and this makes the semantics not verifiable since one cannot access to those mental states to check whether the conditions are really satisfied or not. Consequently, such category of approaches violates one of the important properties of a semantics.

In the second category of semantics, called *social* and developed in [5, 13, 14], primacy is given to the interactions among the agents. The semantics is based on social *commitments* brought about by performing a speech act. For example, by affirming a data, the agent commits on the truth of that data. After a promise, the agent is committed carrying it out. There are several weak points of this approach and the most relevant one is the fact that the concept of commitment itself is ambiguous and its semantics is not clear. According to the performative act, the semantics of the commitment differs. (See section 5 for more details)

Another limitation of both approaches is the fact that they are neither modular nor general. Since they are defined on particular speech acts, if a new speech act is needed for a particular dialogue, then the semantics should be extended.

Our aim is to define a semantics which prevents the shortcomings of the existing approaches while keeping their benefits. Thus, we propose a semantics which satisfies the two main properties. Our approach is social in nature since it is based on the notion of commitments. An agent may have two kinds of commitments:

1. The commitments made by the agent itself during the dialogue such as promises and that it should respect. Such commitments allow the evaluation of the degree of *honesty* of the agent. Indeed, an agent which respects its commitments is said *honest*.
2. The commitments for replying to requests made by the other agents. For instance if the agent receives a question from another agent, then it should answer. Such commitments ensures the *cooperativity* of the agent.

From a syntactic point of view, these commitments are modelled via a notion of *commitment store* as originally defined by MacKenzie in [9]. Indeed, each agent is supposed to be equipped with a commitment store which will keep track of the different commitments of the agent. These stores are accessible for all the agents.

The semantics of a commitment is *unique* and is given by the sanction, called *penalty*, associated to a commitment in case it is not respected by the agent.

Another interesting feature of our semantics is the fact that it is defined on the well-known *categories* of speech acts defined by Searle in [11, 12]. Indeed, Searle has defined five categories of performatives: the *assertives*, the *directives*, the *commissives* and finally the *expressives*. We will show that we can define a semantics for each category without worrying about the different speech acts it may contain. This makes our semantics flexible and more general than the existing ones.

The proposed semantics is also verifiable since the penalty associated to a move is computed directly from its category and the agent commitment store which is visible to all agents; so there is no need to know what the agent really believes, only the category of the moves and previous moves are taken into account.

The paper is organized as follows: Section 2 introduces the logical language that will be used throughout the paper. In section 3 we present the basic concepts of our semantics. In particular, we define the two kinds of commitments of an agents as well as the two corresponding penalties. In section 4, we define the semantics of the four categories of speech acts defined by Searle (we do not consider the expressive category). Section 5 compares our approach to existing semantics and finally, section 6 concludes with some remarks and perspectives.

2 Background

In this section we start by presenting the logical language which will be used throughout this paper. We will distinguish among *action variables* ($\mathcal{AV}$) and *non-actions variables* ($\mathcal{NAV}$). $\mathcal{AV}$ and $\mathcal{NAV}$ are supposed to be disjoint (i.e. $\mathcal{AV} \cap \mathcal{NAV} = \varnothing$).

$\mathcal{L}$ will denote a propositional language built from $\mathcal{AV} \cup \mathcal{NAV}$. $\vdash$ denotes classical inference and $\equiv$ denotes logical equivalence.

$Arg(\mathcal{L})$ will denote the set of all the arguments that can be constructed from $\mathcal{L}$. An *argument* is a pair (Γ, ϕ) where Γ is a set of formulas of $\mathcal{L}$, called the *support* of the argument and $\phi \in \mathcal{L}$ is its *conclusion*. In [6], Dung has presented a powerful argumentation framework which takes as input a set of arguments and the different conflicts which may exist between them, and returns among all the arguments the "good" ones, called the *acceptable* arguments.

Let $\mathcal{A} = \{a_1, \ldots, a_n\}$ denote the set of agents. A communication language is based on a set of *speech acts*. In what follows, $\mathcal{S}$ denotes the set of speech acts of a given communication language. In [2], a communication language has been defined using the following set of speech acts: {*Question*, *Challenge*, *Assert*, *Accept*, *Refuse*, *Request*, *Promise*, *Retract*, *Argue* }. *Assert* allows the exchange of information such as “the weather is beautiful” and “It is my intention to hang a picture”. *Request* is invoked when an agent cannot, or prefer not to, achieve its intentions alone. The proposition requested differs from an asserted proposition in that it cannot be proved true or false. The decision on whether to accept it or not hinges upon the relation it has to the agent’s intentions. An agent will make a *Promise* when it needs to request something from another, and has something it doesn’t need which can offer in return. In replying to an assertion, a request or a promise, an agent can accept, refuse or ask a question such as ”is it the case that p is true ?”. Another kind of question is called a challenge. It allows an agent to ask another agent why it has asserted a proposition or requested something, for example “why newspapers can’t publish the information X?”. The answer to a challenge should be an *argument. Argue* allows agents to exchange arguments, and *Retract* allows them to retract propositions previously asserted or requested.

In [11, 12], Searle has identified five categories of speech acts according to what he calls their illocutionary purpose, their psychological direction to the world, their expressed state, and their propositional content. The five categories are as follows:

Assertive speech acts: The speaker claims that some proposition is true. This category contains speech acts like “inform” or “assert”. For instance, *Inform: the sky is blue* or *Inform: it is my intention to hang a mirror.* In what follows, we will refer to the set of all speech acts of this category by $\mathcal{S}_1$.

Directive speech acts: The speaker attempts to get the hearer to do something. This category includes speech acts like “Request”, “Question” and “Challenge”. For instance, *Request: clean your room!*, *Question: is p true.* In what follows, we will refer to the set of all speech acts of this category by $\mathcal{S}_2$.

Commissive speech acts: The speaker commits to some future course of action. An example of a commissive speech act is “promise”, for instance *Promise: I will do it.* In what follows, we will refer to the set of all speech acts of this category by $\mathcal{S}_3$.

Expressive speech acts: The speaker expresses some psychological state. For instance, an agent can say *Im sorry.*

Declarative speech acts: The speaker brings about a different state of the world. For instance, “Propose”, “Accept”, “Refuse”, “Retract” are declaratives. In what follows, we will refer to the set of all speech acts of this category by $\mathcal{S}_4$.

In this paper, we are more concerned with setting a semantics for dialogues between artificial agents. Thus, the category of expressive speech acts will not be considered here.

$\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4$ is seen as a partition of the set $\mathcal{S}$ of speech acts. This means that $\forall i, j \in [1, 4], i \neq j \Rightarrow \mathcal{S}_i \cap \mathcal{S}_j = \varnothing$ and $\mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3 \cup \mathcal{S}_4 = \mathcal{S}$. Hence, every speech act symbol belongs to one and only one of the four categories $\mathcal{S}_1$ or $\mathcal{S}_2$ or $\mathcal{S}_3$ or $\mathcal{S}_4$.

Definition 1 *A* move *is a formula $a : x$ where $a \in \mathcal{S}$ is a speech act and x is either a propositional formula ($x \in \mathcal{L}$) or an argument ($x \in Arg(\mathcal{L})$). Let $\mathcal{M}$ be the set of moves.*

For instance, $Question : sky\_blue$ is a move meaning that the agent asks whether the sky is blue or not. Here, "Question" is a speech act and sky_blue is a propositional formula (whose value is true or false).

3 The semantics

3.1 Commitment

In the scientific literature, one can find proposals where the semantics of an ACL is defined in terms of commitments. Examples of these are given by Colombetti [5] and Singh [13, 14]. The authors argued that agents are social entities, involved in social interactions, then they are committed to what they say.

In recent years, it has been argued that informal logic has much to offer to the analysis of inter-agent communication. Central in these approaches are the notions of dialogue games and (social) commitments. One rather influential dialogue game is DC, proposed by MacKenzie [9] in the course of analysing the fallacy of question-begging. DC provides a set of rules for arguing about the truth of a proposition. Each player has the goal of convincing the other, and can assert or retract facts, challenge the other player's assertions, ask whether something is true or not, and demand that inconsistencies be resolved.

Associated with each player is a *commitment store*, which holds the commitments of the players during the dialogue. Commitments here are the information given by the players during the dialogue. There are then rules which define how the commitment stores are updated. Take for instance the assertion, it puts a propositional statement in the speaker's commitment store. What this basically means is that, when challenged, the speaker will have to justify his claim. But this does not presuppose that the challenge will come at the next turn in the dialogue.

For our purpose, we adopt this presentation of commitments. In this paper we are not interested in modeling the agent's reasoning, we only consider what is said by each agent. Our purpose is to provide a semantics for the dialogue without worrying about the mental states of the agents.

Each agent is supposed to be equipped with a commitment store, accessible to all agents, which will contain its commitments made during the dialogue. The union of the commitment stores of all agents at turn t can be viewed as the state of the dialogue at turn t. We adopt a commitment store much more structured than the one presented in previous works on dialogue [1, 3]. It keeps tracks of two kinds of commitments:

– The commitments made in the dialogue by the agent itself such as assertions and promises.
– The commitments made by other agents, such as requests, challenges and questions. For instance if an agent a_i makes a request r to another agent a_j, the request (r) is stored in the commitment store of a_j. Hence, a_j is said committed to satisfy this request.

More formally, for each agent a_i, we denote CS_i its commitment store, containing statements and issues to be resolved.

Definition 2 *A* commitment store CS_i *associated to an agent* a_i *is a pair:*

$$CS_i = \langle A_i, O_i \rangle$$

where

– $A_i \subseteq \mathcal{M}$ *stands for the set of moves of agent* a_i *whose speech acts are in* $\{\mathcal{S}_1, \mathcal{S}_3, \mathcal{S}_4\}$.
– $O_i \subseteq \mathcal{M}$ *stands for the set of moves received by agent* a_i *and whose speech acts are in* $\mathcal{S}_2$.

The disctinction between the two kinds of commitments is necessary because statements expressed by the agent itself commit its honesty whereas statements expressed by other agents commit its cooperativity. They are the different by nature and, in our opinion, should not be mixed.
From the above definition, it is also clear that assertive, commissive and declarative speech acts are related to the first kind of commitments (honesty), whereas directive speech acts are related to the second kind of commitments.

3.2 Penalties

Each time an agent makes a speech act, it makes a commitment. So, it is natural to associate to each commitment a penalty which sanctions the agent if it does not respect this commitment. For the sake of simplicity, we consider that the cost associated to a move depends only of its category.

Definition 3 (Penalty) *Let* α_1*,* α_2*,* α_3 *and* α_4 *be numbers in* $[0, +\infty]$ *associated respectively to the categories* $\mathcal{S}_1$*,* $\mathcal{S}_2$*,* $\mathcal{S}_3$ *and* $\mathcal{S}_4$*.*
A penalty *associated with a category* $\mathcal{S}_i$ *is the number* α_i*.*

We distinguish between two kinds of bad behaviors: i) to not be honest, and ii) to not cooperate. Hence, to each commitment store a pair of costs is defined. Formally, the cost associated to the commitment store $CS_i = \langle A_i, O_i \rangle$ of agent a_i at time t, is a pair $\langle c(A_i), c(O_i) \rangle$, where $c(A_i)$ is the cost associated to the statements of agent a_i which are violated, and $c(O_i)$ is the cost associated to the requests made by other agents and to which agent a_i has not answered yet. These costs are computed by summing penalties of moves. Deciding which moves are to take into account depends on their categories.

With such a semantics for an agent communication language, a "rational" protocol should be defined in such a way that each agent aims to minimize the costs associated to its commitment store. This notion of protocol is beyond the scope of this paper.

4 The semantics of the categories of speech acts

In this section we will examine how the penalty associated to a move of each category can be computed, i.e., we will describe when a given commitment is violated by the agent.

4.1 Categories: Assertives ($\mathcal{S}_1$) and Declaratives ($\mathcal{S}_4$)

The assertives and the declaratives behave exactly in the same way, but apply on two distinct languages. Indeed, for the assertives, the content of the moves are formulas of $\mathcal{L}$ built only on the set of non-action variable ($\mathcal{NAV}$), whereas the ones of declaratives are built on action variables ($\mathcal{AV}$).
During a dialogue, an agent may claim that some statement is true. This agent is then, not allowed to contradict itself during all the dialogue otherwise it should pay a penalty.

Storage All the assertive moves are stocked in the commitment store of the agent, exactly in the part A_i. When an assertive move comes as a response to a question or a challenge, for instance, this question or challenge is removed from the set O_i. However, if it is rather a response to a kind of commissive speech act like a promise, then the promise is removed from A_i.
$\forall a \in \mathcal{S}_1$, there are two cases, either the associated assertion is a formula or it is an argument:

- $\forall x \in \mathcal{L}$, if agent a_i commits $a : x$ then
  - $a : x$ is *added* to A_i, and
  - $\forall (a' : y) \in O_i$, such that $a' \in \mathcal{S}_2$ and $y \in \mathcal{L}$ and $x \vdash y$, $a' : y$ is *removed* from O_i, and
  - $\forall (a' : y) \in A_i$, such that $a' \in \mathcal{S}_3$ and $y \in \mathcal{L}$ and $x \vdash y$, $a' : y$ is *removed* from A_i.
- $\forall x = \langle \Gamma, \varphi \rangle \in Arg(\mathcal{L})$, if agent a_i commits $a : x$ then
  - $a : x$ is *added* to A_i, and
  - $\forall (a' : y) \in O_i$, such that $a' \in \mathcal{S}_2$ and $y \in \mathcal{L}$ and $\varphi \vdash y$, $a' : y$ is *removed* from O_i, and
  - $\forall (a' : y) \in A_i$, such that $a' \in \mathcal{S}_3$ and $y \in \mathcal{L}$ and $\varphi \vdash y$, $a' : y$ is *removed* from A_i

Associated Cost: Costs associated to assertives concern non-honesty. An *assertive* move $a : x$, where $a \in \mathcal{S}_1$ and $x \in \mathcal{L} \cup Arg(\mathcal{L})$, is a *violated commitment* if the part A_i of agent a_i's commitments store allows to deduce that the formula x is false or to deduce that one formula of the support of the argument x is false. Hence, the cost associated to an assertive is given by:
$\forall (a : x) \in A_i$, if $a \in \mathcal{S}_1$, there are two cases, either the assertion x is a formula or it is an argument:

- $\forall x \in \mathcal{L}$, $c(a : x) = \begin{cases} \alpha_1 \text{ if } A_i \vdash \neg x, \\ 0 \quad \text{otherwise} \end{cases}$
- $\forall x = \langle \Gamma, \varphi \rangle \in Arg(\mathcal{L})$, $c(a : x) = \begin{cases} \alpha_1 \text{ if } \exists \varphi \in \Gamma, A_i \vdash \neg \varphi, \\ 0 \quad \text{otherwise} \end{cases}$

4.2 Category $\mathcal{S}_2$: Directives

The agent attempts to get another agent to do something. For instance, Request, Question, Challenge are Directives.

Storage Directives of Agent a_i towards Agent a_j are stored in the part O_j of the commitment store of agent a_j, if there is not already an answer to it in A_j (which can be either assertive or declarative):
$\forall a \in \mathcal{S}_2$, the directive $a : x$ of Agent a_i towards Agent a_j is *added* to O_j if $\nexists (a' : y) \in A_j$, such that $a' \in \mathcal{S}_1 \cup \mathcal{S}_4$ and:

- $y \in \mathcal{L}$ and $y \vdash x$
- or $y = \langle \Gamma, \varphi \rangle \in Arg(\mathcal{L})$ and $\varphi \vdash y$.

Associated Cost Costs associated to directives are only penalties of non-cooperativity. It means that a directive is violated if it is in the part O_i of Agent a_i (agent a_i has not already answered to it).
$\forall (a : x) \in O_i$ if $a \in \mathcal{S}_2, \forall x \in \mathcal{L}$, $c(a : x) = \alpha_2$

4.3 Category $\mathcal{S}_3$: Commissives

The agent commits to some future course of action. For instance, Promise is in the Commissives. The most general form of promise is "if φ is true then I swear that ψ will be true", "if you do that, I will kill you". It can be encoded by the move $Promise : \varphi \to \psi$.

Storage Commissives of Agent a_i are stored in the part A_i of its commitment store if the associated formula is not already true in A_i (either because there is an assertive or a declarative move which makes it true):
$\forall a \in \mathcal{S}_3$, the commissive $a : x$ of Agent a_i is *added* to A_i if $\nexists (a' : y) \in A_i$, such that $a' \in \mathcal{S}_1 \cup \mathcal{S}_4$ and:

- $y \in \mathcal{L}$ and $y \vdash x$
- or $y = \langle \Gamma, \varphi \rangle \in Arg(\mathcal{L})$ and $\varphi \vdash y$.

Associated Cost Costs associated to commissives are only costs of non-honesty. As long as a commissive is in the commitment store, it costs a penalty since it means that this commissive has not been made.
$\forall (a:x) \in A_i$ If $a \in \mathcal{S}_3, \forall x \in \mathcal{L}$, $c(a:x) = \alpha_3$

4.4 Summary

To sum up, the penalties associated to a commitment store CS_i of an agent a_i at turn t are:

cooperation penalty:

$$p_c(CS_i) = \sum_{(a:x) \in O_i} \alpha_2$$

honesty penalty:

$$p_h(CS_i) = \sum_{(a:x)\in A_i \quad s.t. \quad a \in \mathcal{S}_1 (resp. \mathcal{S}_4) \quad and \quad x = \langle \Gamma, \varphi \rangle \in Arg(\mathcal{L}) \text{ and } \exists \psi \in \Gamma \text{ s.t. } A_i \vdash \neg\psi} \alpha_1 (resp. \alpha_4)$$

$$+ \sum_{(a:x)\in A_i \quad s.t. \quad a \in \mathcal{S}_1 (resp. \mathcal{S}_4) \quad and \quad x \in \mathcal{L} \text{ and } A_i \vdash \neg x} \alpha_1 (resp. \alpha_4)$$

$$+ \sum_{(a:x)\in A_i \quad s.t. \quad a \in \mathcal{S}_3} \alpha_3$$

Or equivalently, (since we suppose that $c(a:x)$ only depends on the category of a):

$$p_c(CS_i) = \sum_{(a:x)\in O_i, a \in \mathcal{S}_2} c(a:x)$$

and

$$p_h(CS_i) = \sum_{(a:x)\in A_i \begin{cases} s.t. \quad a \in \mathcal{S}_1 \cup \mathcal{S}_4 \\ or \quad and \begin{cases} x = \langle \Gamma, \varphi \rangle \in Arg(\mathcal{L}) \text{ and } \exists \psi \in \Gamma \text{ s.t. } A_i \vdash \neg\psi \\ or \\ x \in \mathcal{L} \text{ and } A_i \vdash \neg x \end{cases} \\ s.t. \quad a \in \mathcal{S}_3 \end{cases}} c(a:x)$$

5 Related work

The first standard agent communication language is KQML [7]. It has been developed within the Knowledge Sharing Effort, a vast research program funded by DARPA (the US Defense Advanced Research Projects Agency). More recently, the Foundation for Intelligent Physical Agents (FIPA) has proposed a new standard, named ACL [8]. Both KQML and ACL have been given a mentalistics semantics. The semantics is based on a notion of speech act close to the concept of illocutionary act as developed in speech act theory [4, 12].
Such semantics assumes, more or less explicitly, some underlying hypothesis in particular, that the agents are sincere and cooperative. While this may be well fitted for some special cases of interactions, it is obvious that some dialogue types listed by Walton and Krabbe in [15] are not cooperative. For example, assuming sincerity and cooperativity in negotiation may lead to poor negotiators. Another more important limitation of this approach is the fact that it is not verifiable since it is based on the mental states of the agents. Our semantics does not refer at all to the mental states of the agents. Moreover, it does not treat particular speech acts as it is the case with this mentalistic approach.

In the second approach, called *social* and developed in [5, 13, 14], primacy is given to the interactions among the agents. The semantics is based on social *commitments* brought about by performing a speech act. For example, by affirming a data, the agent commits on the truth of that data. After a promise, the agent is committed carrying it out. There are several weak points of this approach and we summarize them in the three following points:
1. The definition of commitments complicates the agent architecture in the sense that it needs an ad hoc apparatus. The commitments are introduced especially for modeling communication. Thus agents should reason not only on their beliefs, etc but also on the commitments. In our approach, we didn't introduce any new language to treat commitments. We call a commitment any information stocked in a commitment store. Handling these commitments (to add a new commitment, to retract a commitment, to achieve a commitment, to violate a commitment) is done directly on the commitment store.
2. The level at which communication is treated is very abstract, and there is a considerable gap to fill in order to bring the model down to the level of implementation. However, the semantics presented in this paper can be implemented easily.
3. The concept of commitment is ambiguous and its semantics is not clear. According to the speech act, the semantics of the commitment differs. For example:

Inform: by affirming a data, the agent commits on the truth of that data. The meaning of the commitment here is not clear. It may be that the agent can justify the data or can defend it against any attack, or the agent is sincere.

Request: According to Colombetti [5] after a request, the receiver precommits to carry it out. This idea is in our opinion drawn from the notion of protocol. The protocol specifies for each act the set of allowed replies. So after a request, generally the receiver can accept it, reject it or propose another alternative.

The last approach developed by Pitt and Mandani in [10] is based on the notion of protocol. A protocol defines what sequences of moves are conventionally expected in the dialogue. The meaning of a speech act then equates with the set of the possible following answers.
However, protocols are often technically finite state machines. This turns out to be too rigid in several circumstances. Current research aims at defining flexible protocols, which rely more on the *state of the dialogue*, and less on dialogue history. This state of dialogue is captured through the notion of commitment.

6 Conclusion

This paper has introduced a *general* semantics of any agent communication language. The semantics is general in the sense that it can be used with any set of speech acts since it is defined on categories of speech acts rather than the speech acts themselves. Indeed, for the first time, we have a semantics of an ACL which is defined independently from the syntax of that ACL. This makes the semantics *flexible* and general.
The new semantics is social in nature and is based on the notion of *commitments*. However, our modeling of this notion is very different from the existing approaches. Indeed, the commitments are modeled as in MacKenzie's system DC [9]. A commitment is any information stocked in the *commitment store* of the agent. Two kinds of commitments have been distinguished: the commitments to satisfy or to respect what the agent has already said, promised, etc. and the commitments to answer to other agents.
Unlike in the existing approaches, the semantics of a commitment is *unique* and is given via the notion of penalty that should be paid if an agent does not respect that commitment.
Our semantics goes somewhat beyond the existing approaches in giving an operational, verifiable and flexible semantics. An extension of this work would be to simplify the protocols by extending our semantics by rules which were generally defined in the protocol itself. For instance, in the semantics, we can sanction agents which repeat, the same move several times during a dialogue. Another extension would be to refine the penalties by introducing granularity in the categories of speech acts. The idea is, maybe, to associate different penalties for the speech acts in order to capture the idea that some commitments are harder to violate than others. The same idea applies also to the content of the moves.

References

1. L. Amgoud, N. Maudet, and S. Parsons. Modelling dialogues using argumentation. In E. Durfee, editor, *Proceedings of the 4th International Conference on Multi-Agent Systems (ICMAS00)*, pages 31–38, Boston, USA, July 2000. IEEE Press.

2. L. Amgoud, N. Maudet, and S. Parsons. An argumentation-based semantics for agent communication languages. In F. Van Harmelen, editor, *Proceedings of the European Conference on Artificial Intelligence (ECAI-2002)*, pages 38–42, Lyon, France, July 2002. IOS Press.
3. L. Amgoud, S. Parsons, and N. Maudet. Arguments, dialogue, and negotiation. In W. Horn, editor, *Proceedings of the European Conference on Artificial Intelligence (ECAI-2000)*, pages 338–342, Berlin, Germany, August 2000. IOS Press.
4. J. L. Austin. How to do things with words. In *Clarendon Press, Oxford Uk*, 1962.
5. M. Colombetti. A commitment-based approach to agent speech acts and conversations. In *Proceedings of the Workshop on Agent Languages and Conversation Policies. 14th International Conference on Autonomous Agents*, pages 21–29, 2000.
6. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77:321–357, 1995.
7. T. Finin, Y. Labrou, and J. Mayfield. Kqml as an agent communication language. In *J. Bradshaw, ed. Softaware agents, MIT Press, Cambridge.*, 1995.
8. FIPA. Agent communication language. In *FIPA 97 Specification, Foundation for Intelligent Physical Agents.*, 1997.
9. J. MacKenzie. Question-begging in non-cumulative systems. *Journal of philosophical logic*, 8:117–133, 1979.
10. J. Pitt and A. Mamdani. A protocol based semantics for an agent communication language. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 486–491, 1999.
11. J. R. Searle. Speech acts. In *Cambridge University Press*, 1969.
12. J. R. Searle and D. Vanderveken. Foundations of illocutionary logic. In *Cambridge University Press*, 1985.
13. M. P. Singh. Agent communication languages: Rethinking the principles. In *IEEE Computer*, pages 40–47, 1998.
14. M. P. Singh. A social semantics for agent communication languages. In *IJCAI'99 Workshop on Agent Communication Languages*, pages 75–88, 1999.
15. D. N. Walton and E. C. W. Krabbe. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning.* State University of New York Press, Albany, NY, 1995.

Reasoning about Epistemic States of Agents by Modal Logic Programming

Linh Anh Nguyen

Institute of Informatics, University of Warsaw
ul. Banacha 2, 02-097 Warsaw, Poland
nguyen@mimuw.edu.pl

Abstract. Modal logic programming is one of appropriate approaches to deal with reasoning about epistemic states of agents. We specify here the least model semantics, the fixpoint semantics, and an SLD-resolution calculus for modal logic programs in the multimodal logic $KD4I_g5_a$, which is intended for reasoning about belief and common belief of agents. We prove that the presented SLD-resolution calculus is sound and complete. We also present a formalization of the wise men puzzle using a modal logic program in $KD4I_g5_a$. This shows that it is worth to study modal logic programming for multi-agent systems.

1 Introduction

Reasoning is an important aspect of agents. In order to be able to make right actions, an agent should have general knowledge of the field it works on, information about the environment, and abilities to interact with the environment, to make inferences, and to revise its knowledge base. In multi-agent systems, agents should be able to communicate, collaborate, and sometimes compete with each other. For this aim, an agent should have knowledge about other agents in the system and be able to reason about their epistemic states. It is not that an agent can have all information it wants or can reason exactly as the others, but at least it can simulate epistemic states of the other agents, using some assumptions. The wise men puzzle introduced by McCarthy [17] is an example of reasoning about epistemic states of agents. We will study it in Section 3.

Modal logics and logic programming are useful instruments for multi-agent systems. Using modal logics is a natural way to represent and reason about knowledge and belief of agents (see, e.g., [10, 28, 27, 14, 7, 1]). Logic programming is also useful because logical implication is probably the inference form humans use most and want to adopt for multi-agent systems. Thus, one can think about modal logic programming as an approach to deal with reasoning about epistemic states of agents.

Modal logic programming has been studied in a number of works (see the earlier surveys [24, 12] and the later works [23, 5, 19, 22]). There are two approaches: the direct approach [11, 3, 5, 19, 22] and the translation approach [8, 23]. The first approach directly uses modalities, while the second one translates modal logic programs to classical logic programs. In this paper we will use the direct approach. This approach is justifiable, as the direct approach deals with modalities more closely, and modalities allow us to separate object-level and epistemic-level notions nicely.

In [19], we developed a fixpoint semantics, the least model semantics, and an SLD-resolution calculus in a direct way for modal logic programs in all of the basic serial monomodal logics. In that work we do not assume any special restriction on occurrences of $\Box$ and $\Diamond$ in programs and goals. In [22], we generalized the methods of [19] and gave a general framework for developing fixpoint semantics, the least model semantics, and SLD-resolution calculi for logic programs in normal multimodal logics whose frame restrictions consist of the conditions of seriality and some classical first-order Horn formulas.

In this work, we instantiate the above mentioned framework for the multimodal logic $KD4I_g5_a$, which was introduced in [20] for reasoning about belief and common belief. We prove that the obtained SLD-resolution calculus is sound and complete. We also give a purely logical formalization of the wise men puzzle using a modal logic program in $KD4I_g5_a$.

The rest of this paper is structured as follows. In Section 2, we give definitions for multimodal logics, define the multimodal logic $KD4I_g5_a$ and the modal logic programming language MProlog. In Section 3, we recall the wise men puzzle and formalize it by an MProlog program in $KD4I_g5_a$. In Section 4, we instantiate the framework given in [22] for $KD4I_g5_a$ in order to specify the least model semantics, the fixpoint semantics, and an SLD-resolution calculus for MProlog programs in $KD4I_g5_a$. Soundness and completeness of the obtained SLD-resolution calculus is proved in Section 5. (Due to the lack of space we do not present proofs involving with the fixpoint semantics and the least model semantics.) Finally, Section 6 contains some concluding remarks.

2 Preliminaries

2.1 Syntax and Semantics of Quantified Multimodal Logics

A language for quantified multimodal logics is an extension of the language of classical predicate logic with modal operators $\Box_i$ and $\Diamond_i$, for $1 \leq i \leq m$ (where m is fixed). The modal operators $\Box_i$ and $\Diamond_i$ can take various meanings. For example, $\Box_i$ can stand for "the agent i believes" and $\Diamond_i$ for "it is considered possible by agent i". The operators $\Box_i$ are called universal modal operators, while $\Diamond_i$ are called existential modal operators. Terms and formulas are defined in the usual way, with an emphasis that if φ is a formula then $\Box_i\varphi$ and $\Diamond_i\varphi$ are also formulas.

A *Kripke frame* is a tuple $\langle W, \tau, R_1, \ldots, R_m \rangle$, where W is a nonempty set of possible worlds, $\tau \in W$ is the *actual world*, and R_i is a binary relation on W, called the *accessibility relation* for the modal operators $\Box_i$, $\Diamond_i$. If $R_i(w, u)$ holds then we say that the world u is accessible from the world w via R_i.

A *fixed-domain Kripke model with rigid terms*, hereafter simply called a Kripke model or just a model, is a tuple $M = \langle D, W, \tau, R_1, \ldots, R_m, \pi \rangle$, where

D is a set called the *domain*, $\langle W, \tau, R_1, \ldots, R_m \rangle$ is a Kripke frame, and π is an interpretation of constant symbols, function symbols and predicate symbols. For a constant symbol a, $\pi(a)$ is an element of D, denoted by a^M. For an n-ary function symbol f, $\pi(f)$ is a function from D^n to D, denoted by f^M. For an n-ary predicate symbol p and a world $w \in W$, $\pi(w)(p)$ is an n-ary relation on D, denoted by $p^{M,w}$.

A *model graph* is a tuple $\langle W, \tau, R_1, \ldots, R_m, H \rangle$, where $\langle W, \tau, R_1, \ldots, R_m \rangle$ is a Kripke frame and H is a function that maps each world of W to a set of formulas.

Every model graph $\langle W, \tau, R_1, \ldots, R_m, H \rangle$ corresponds to a Herbrand model $M = \langle \mathcal{U}, W, \tau, R_1, \ldots, R_m, \pi \rangle$ specified by: $\mathcal{U}$ is the Herbrand universe (i.e. the set of all ground terms), $c^M = c$, $f^M(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$, and $((t_1, \ldots, t_n) \in p^{M,w}) \equiv (p(t_1, \ldots, t_n) \in H(w))$, where $t_1, \ldots, t_n$ are ground terms. We will sometimes treat a model graph as its corresponding model.

A *variable assignment* V w.r.t. a Kripke model M is a function that maps each variable to an element of the domain of M. The value of $t^M[V]$ for a term t is defined as usual.

Given some Kripke model $M = \langle D, W, \tau, R_1, \ldots, R_m, \pi \rangle$, some variable assignment V, and some world $w \in W$, the *satisfaction relation* $M, V, w \vDash \psi$ for a formula ψ is defined as follows:

$$\begin{array}{lll}
M, V, w \vDash p(t_1, \ldots, t_n) & \text{iff} & (t_1^M[V], \ldots, t_n^M[V]) \in p^{M,w}; \\
M, V, w \vDash \Box_i \varphi & \text{iff} & \text{for all } v \in W \text{ such that } R_i(w, v),\ M, V, v \vDash \varphi; \\
M, V, w \vDash \forall x.\varphi & \text{iff} & \text{for all } a \in D,\ (M, V', w \vDash \varphi), \\
 & & \text{where } V'(x) = a \text{ and } V'(y) = V(y) \text{ for } y \neq x;
\end{array}$$

and as usual for other cases (treating $\Diamond_i \varphi$ as $\neg\Box_i\neg\varphi$, and $\exists x.\varphi$ as $\neg\forall x.\neg\varphi$). We say that M satisfies φ, or φ is true in M, and write $M \vDash \varphi$, if $M, V, \tau \vDash \varphi$ for every V. For a set Γ of formulas, we call M a model of Γ and write $M \vDash \Gamma$ if $M \vDash \varphi$ for every $\varphi \in \Gamma$.

If as the class of admissible interpretations we take the class of all Kripke models (with no restrictions on the accessibility relations) then we obtain a quantified multimodal logic which has a standard Hilbert-style axiomatization denoted by K_m. Other *normal (multi)modal logics* are obtained by adding certain axioms to K_m. Mostly used axioms are ones that correspond to a certain restriction on the Kripke frame defined by a classical first-order formula using the accessibility relations. For example, the axiom $(D) : \Box_i\varphi \to \Diamond_i\varphi$ corresponds to the frame restriction $\forall x\, \exists y\, R_i(x, y)$.

For a normal modal logic L whose class of admissible interpretations can be characterized by classical first-order formulas of the accessibility relations, we call such formulas *L-frame restrictions*, and call frames with such properties *L-frames*. We call a model M with an L-frame an *L-model*. We say that φ is *L-satisfiable* if there exists an L-model of φ, i.e. an L-model satisfying φ. A formula φ is said to be *L-valid* and called an *L-tautology* if φ is true in every L-model. For a set Γ of formulas, we write $\Gamma \vDash_L \varphi$ and call φ a *logical consequence* of Γ in L if φ is true in every L-model of Γ.

2.2 The Multimodal Logic $KD4I_g5_a$

Suppose that there are n agents and $m = 2^n - 1$. Let g be an one-to-one function that maps every natural number less than or equal to m to a non-empty subset of $\{1, \ldots, n\}$. Suppose that an index $1 \leq i \leq m$ stands for the group of agents whose indices form the set $g(i)$. To capture belief and common belief of agents, we can extend K_m with the following axioms

- $(D) : \Box_i\varphi \to \neg\Box_i\neg\varphi$ (belief is consistent),
- $(4) : \Box_i\varphi \to \Box_i\Box_i\varphi$ (belief satisfies positive introspection),
- $(I_g) : \Box_i\varphi \to \Box_j\varphi$ if $g(i) \supset g(j)$ (if i indicates a supergroup of a group j then every common belief of i is also a common belief of j).
- $(5_a) : \neg\Box_i\varphi \to \Box_i\neg\Box_i\varphi$ if $g(i)$ is a singleton (belief of a single agent satisfies negative introspection).

Thus, for reasoning about belief and common belief, we can use:

$$KD4I_g5_a = K_m + (D) + (4) + (I_g) + (5_a)$$

Here we want to catch the most important properties of belief and common belief, and the aim is not to give an exact formulation of belief or common belief. The logic $KD4I_g5_a$ was introduced in [20][1]. It is different in the nature from the well-known multimodal logic of common knowledge. It also differs from the modal logic with mutual belief [1].

The given axioms correspond to the following frame restrictions:

Axiom	Corresponding Condition
(D)	$\forall u\, \exists v\, R_i(u, v)$
(4)	$\forall u, v, w\, (R_i(u, v) \land R_i(v, w) \to R_i(u, w))$
(I_g)	$R_j \subseteq R_i$ if $g(i) \supset g(j)$
(5_a)	$\forall u, v, w\, (R_i(u, v) \land R_i(u, w) \to R_i(w, v))$ if $g(i)$ is a singleton

For further reading on epistemic logics, see, e.g., [10, 28, 7, 1].

2.3 Modal Logic Programs

A *modality* is a (possibly empty) sequence of modal operators. A *universal modality* is a modality which contains only universal modal operators. We use $\triangle$ to denote a modality and $\boxdot$ to denote a universal modality. Similarly as in classical logic programming, we use a clausal form $\boxdot(\varphi \leftarrow \psi_1, \ldots, \psi_n)$ to denote the formula $\forall(\boxdot(\varphi \lor \neg\psi_1 \ldots \lor \neg\psi_n))$. We use E to denote a classical atom and A, $B_1, \ldots, B_n$ to denote formulas of the form E, $\Box_i E$, or $\Diamond_i E$.

A *program clause* is a formula of the form $\boxdot(A \leftarrow B_1, \ldots, B_n)$, where $n \geq 0$. $\boxdot$ is called the *modal context*, A the *head*, and $B_1, \ldots, B_n$ the *body* of the program clause. An *MProlog program* is a finite set of program clauses.

[1] The propositional version of $KD4I_g5_a$ is decidable. However, we do not have complexity result for it yet. Hopefully, it will be available in the next version of this paper.

An *MProlog goal atom* is a formula of the form $\boxbox E$ or $\boxbox\Diamond_i E$. An *MProlog goal* is a formula written in the clausal form $\leftarrow \alpha_1, \ldots, \alpha_k$, where each α_i is an MProlog goal atom. The *empty goal* (i.e. the *empty clause*) is denoted by $\diamond$.

In $KD4I_g5_a$, if $g(i)$ is a singleton then we have the equivalence $\nabla_i\nabla'_i\varphi \equiv \nabla'_i\varphi$ for any modal operators ∇_i and ∇'_i with the same modal index i. For this reason, we adopt some restrictions to simplify the form of MProlog programs and goals in $KD4I_g5_a$. An MProlog program is called a *$KD4I_g5_a$-MProlog program* if the modal contexts of its program clauses do not contain subsequences of the form $\Box_i\Box_i$ if $g(i)$ is a singleton. An MProlog goal is called a *$KD4I_g5_a$-MProlog goal* if each of its goal atoms $\triangle E$ satisfies the condition that $\triangle$ does not contain subsequences of the form $\Box_i\Box_i$ or $\Box_i\Diamond_i$ if $g(i)$ is a singleton.

Let P be an $KD4I_g5_a$-MProlog program and $G = \leftarrow \alpha_1, \ldots, \alpha_k$ be an $KD4I_g5_a$-MProlog goal. An *answer* θ for $P \cup \{G\}$ is a substitution whose domain is the set of all variables of G. We say that θ is a *correct answer* in $KD4I_g5_a$ for $P \cup \{G\}$ if θ is an answer for $P \cup \{G\}$ and $P \vDash_{KD4I_g5_a} \forall((\alpha_1 \wedge \ldots \wedge \alpha_k)\theta)$.

It is shown in [20] that MProlog has the same expressiveness power as the general Horn fragment in normal modal logics. Moreover, the restrictions adopted for $KD4I_g5_a$-MProlog do not reduce expressiveness of the language (see [20]).

3 The Wise Men Puzzle

Before considering technical details of semantics of $KD4I_g5_a$-MProlog, we give a formalization of the three wise men puzzle in MProlog. The puzzle is a famous benchmark introduced by McCarthy [17] for AI. It can be stated as follows (cf. [15]). A king wishes to know whether his three advisors (A, B, C) are as wise as they claim to be. Three chairs are lined up, all facing the same direction, with one behind the other. The wise men are instructed to sit down in the order A, B, C. Each of the men can see the backs of the men sitting before them (e.g. C can see A and B). The king informs the wise men that he has three cards, all of which are either black or white, at least one of which is white. He places one card, face up, behind each of the three wise men, explaining that each wise man must determine the color of his own card. Each wise man must announce the color of his own card as soon as he knows what it is. All know that this will happen. The room is silent; then, after a while, wise man A says "My card is white!".

The wise men puzzle has been previously studied in several works (e.g., [17, 15, 9, 6, 2, 23, 4]). McCarthy [17] directly used possible worlds to formalize the puzzle. Konolige [15], Nonnengart [23], and Baldoni [4] also used modal logics for the puzzle. Konolige [15] focused on limited reasoning, Nonnengart [23] used semi-functional translation for modal logic programming, and Baldoni [4] used a prefixed tableau system. Both McCarthy [17] and Nonnengart [23] used some feature of mutual belief, but they did not define it purely. Baldoni [4] adopted too strong versions of axioms 4 and 5, which are not suitable for the puzzle. As other approaches for the wise men puzzle, Elgot-Drapkin [9] used step-logics, while Cimatti and Serafini [6], Attardi and Simi [2] studied reasoning in belief-contexts. Our formalization of the wise men puzzle given below uses $KD4I_g5_a$-MProlog. It is more elegant than the above-mentioned formalizations, as it uses a modal logic with a clear semantics of common belief in a direct way.

As reported in [21], we have designed and implemented a modal logic programming system, also called MProlog. In that system, SLD-resolution calculi for MProlog can be specified according to the theoretical framework given in [22]. An instantiation of that framework for $KD4I_g5_a$ is presented in the next section. Its implementation (of SLD-resolution) is denoted by ccKD4Ig5a. In that implementation, *bel* denotes belief and *pos* denotes possibility, and modalities are represented by lists, e.g. $\Box_i\langle X\rangle_j\Diamond_k q(a)$ is represented by $[bel(I), pos(J,X), pos(K)] : q(a)$. The implemented calculus requires definitions of predicates $singleton\_group/1$, $subgroup/2$, and $union\_group/3$. Denote the wise men by a, b, c, and the possible groups by gAB, gAC, gBC, $gABC$, where, e.g., $gABC = \{a, b, c\}$. Thus, $[bel(gABC)] : \varphi$ means that φ is a common belief of the group $\{a, b, c\}$. Define the mentioned required predicates in the usual way. The three wise men problem can be formalized by the following program:

```
:- calculus ccKD4Ig5a.
% If Y sits behinds X then X's card is white if Y considers this as possible.
[bel(gABC)]: (white(X) :-
        member(X, [a,b,c]), member(Y, [a,b,c]), X @< Y, [pos(Y)]:white(X)).
% The following formula is "dual" to the above formula.
[bel(gABC)]: ([bel(Y)]:black(X) :-
        member(X, [a,b,c]), member(Y, [a,b,c]), X @< Y, black(X)).
% At least one of the wise men has a white card.
[bel(gABC)]: (white(a) :- black(b), black(c)).
[bel(gABC)]: (white(b) :- black(c), black(a)).
[bel(gABC)]: (white(c) :- black(a), black(b)).
/* Each of B and C does not know the color of his own card. In particular, each
of the men considers that it is possible that his own card is black. */
[bel(gABC),pos(b)]:black(b).
[bel(gABC),pos(c)]:black(c).
```

The question is whether A believes that his card is white. It is passed to the interpreter as $mcall([bel(a)] : white(a))$ and solved in less than 1 second[2] using certain option settings.

The above program uses the syntax of the implemented system. We give below a version using the purely logical formalism of MProlog. For clarity, instead of numeric indices we use a, b, c, ab, ac, bc, abc with the meaning that $g(a) = \{a\}$, $g(b) = \{b\}$, $g(c) = \{c\}$, ..., and $g(abc) = \{a, b, c\}$. Let P_{wise_men} be the following program:

$$\begin{aligned}\varphi_1 &= \Box_{abc}\,(white(a) \leftarrow \Diamond_b\, white(a))\\ \varphi_2 &= \Box_{abc}\,(white(a) \leftarrow \Diamond_c\, white(a))\end{aligned}$$

[2] on TravelMate 230X, 1.7GHz-M

$$\begin{array}{lcl}
\varphi_3 &=& \Box_{abc}\,(white(b) \leftarrow \Diamond_c\, white(b)) \\
\varphi_4 &=& \Box_{abc}\,(\Box_b\, black(a) \leftarrow black(a)) \\
\varphi_5 &=& \Box_{abc}\,(\Box_c\, black(a) \leftarrow black(a)) \\
\varphi_6 &=& \Box_{abc}\,(\Box_c\, black(b) \leftarrow black(b)) \\
\varphi_7 &=& \Box_{abc}\,(white(a) \leftarrow black(b), black(c)) \\
\varphi_8 &=& \Box_{abc}\,(white(b) \leftarrow black(c), black(a)) \\
\varphi_9 &=& \Box_{abc}\,(white(c) \leftarrow black(a), black(b)) \\
\varphi_{10} &=& \Box_{abc}\Diamond_b\, black(b) \\
\varphi_{11} &=& \Box_{abc}\Diamond_c\, black(c)
\end{array}$$

The goal is $\leftarrow \Box_a white(a)$. We will continue this example in Section 4.5. For a formalization of the puzzle with n wise men, see [22].

4 Semantics of $KD4I_g5_a$-MProlog Programs

In this section, we present the least model semantics, the fixpoint semantics and an SLD-resolution calculus for $KD4I_g5_a$-MProlog programs. For abbreviation, *from now on we use L to denote $KD4I_g5_a$.*

4.1 Labeled Modal Operators

When applying the direct consequence operator $T_{L,P}$ for an MProlog program P in L, if we obtain an "atom" of the form $\triangle\Diamond_i E$, then to simplify the task we label the modal operator $\Diamond_i$. Labeling allows us to address the chosen world(s) in which this particular E must hold. A natural way is to label $\Diamond_i$ by E to obtain $\langle E\rangle_i$. On the other hand, when dealing with SLD-derivation, we cannot change a goal $\leftarrow \Diamond_i(A \wedge B)$ to $\leftarrow \Diamond_i A, \Diamond_i B$. But if we label the operator $\Diamond_i$, let's say by X, then we can safely change $\leftarrow \langle X\rangle_i(A \wedge B)$ to $\leftarrow \langle X\rangle_i A, \langle X\rangle_i B$.

We will use the following notations:

- $\top$: the *truth* symbol, with the usual semantics[3];
- E, F : classical atoms (which may contain variables) or $\top$;
- X, Y, Z : variables for classical atoms or $\top$, called *atom variables*;
- $\langle E\rangle_i$, $\langle X\rangle_i$: $\Diamond_i$ labeled by E or X;
- ∇ : $\Box_i$, $\Diamond_i$, $\langle E\rangle_i$, or $\langle X\rangle_i$, called a modal operator;
- $\triangle$: a (possibly empty) sequence of modal operators, called a *modality*;
- $\boxdot$: a *universal modality*;
- A, B : formulas of the form E or ∇E, called *simple atoms*;
- α, β : formulas of the form $\triangle E$, called *atoms*;
- φ, ψ : *(labeled) formulas* (i.e. formulas that may contain $\langle E\rangle_i$ and $\langle X\rangle_i$).

We use subscripts beside ∇ to indicate modal indexes in the same way as for $\Box$ and $\Diamond$. To distinguish a number of modal operators we use superscripts of the form (i), e.g. $\Box^{(1)}$, $\Box^{(2)}$, $\nabla^{(i)}$, $\nabla^{(i')}$.

[3] i.e. it is always true that $M, V, w \models \top$

A *ground formula* is a formula with no variables and no atom variables. A modal operator is said to be *ground* if it is $\Box_i$, $\Diamond_i$, or $\langle E \rangle_i$ with E being $\top$ or a ground classical atom. A *ground modality* is a modality that contains only ground modal operators. A *labeled modal operator* is a modal operator of the form $\langle E \rangle_i$ or $\langle X \rangle_i$.

Denote $EdgeLabels = \{\langle E \rangle_i \mid E \in \mathcal{B} \cup \{\top\}$ and $1 \leq i \leq m\}$, where $\mathcal{B}$ is the Herbrand base (i.e. the set of all ground classical atoms). The semantics of $\langle E \rangle_i \in EdgeLabels$ is specified as follows. Let $M = \langle D, W, \tau, R_1, \ldots, R_m, \pi \rangle$ be a Kripke model. A *$\Diamond$-realization function on M* is a partial function $\sigma : W \times EdgeLabels \to W$ such that if $\sigma(w, \langle E \rangle_i) = u$, then $R_i(w, u)$ holds and $M, u \vDash E$. Given a $\Diamond$-realization function σ, a world $w \in W$, and a ground formula φ, the satisfaction relation $M, \sigma, w \vDash \varphi$ is defined in the usual way, except that $M, \sigma, w \vDash \langle E \rangle_i \psi$ iff $\sigma(w, \langle E \rangle_i)$ is defined and $M, \sigma, \sigma(w, \langle E \rangle_i) \vDash \psi$. We write $M, \sigma \vDash \varphi$ to denote that $M, \sigma, \tau \vDash \varphi$. For a set I of ground atoms, we write $M, \sigma \vDash I$ to denote that $M, \sigma \vDash \alpha$ for all $\alpha \in I$; we write $M \vDash I$ and call M a model of I if $M, \sigma \vDash I$ for *some* σ.

4.2 Model Generators

We define that a modality $\nabla^{(1)}_{i_1} \ldots \nabla^{(k)}_{i_k}$ is in the *L-normal form* if for all $1 \leq j < k$ if $g(i_j)$ is a singleton then $i_j \neq i_{j+1}$. (Note that if $g(i)$ is a singleton then $\nabla_i \nabla'_i \varphi \equiv \nabla'_i \varphi$ is $KD4I_g5_a$-valid.) A modality is in *L-normal labeled form* if it is in L-normal form and does not contain modal operators of the form $\Diamond_i$ or $\langle \top \rangle_i$. An atom is in *L-normal (labeled) form* if it is of the form $\triangle E$ with $\triangle$ in L-normal (labeled) form. An atom is in *almost L-normal labeled form* if it is of the form $\triangle A$ with $\triangle$ in L-normal labeled form.

A *model generator* is a set of ground atoms not containing $\Diamond_i$, $\langle \top \rangle_i$, $\top$. An *L-normal model generator* is a model generator consisting of atoms in L-normal labeled form.

We will define the *standard L-model* of an L-normal model generator I so that it is a *least L-model* of I (where a model M is *less than or equal to* a model M' if for every positive ground formula φ without labeled operators, if $M \vDash \varphi$ then $M' \vDash \varphi$). In the construction we will use the operator Ext_L defined below.

A *forward rule* is a schema of the form $\alpha \to \beta$, while a *backward rule* is a schema of the form $\alpha \leftarrow \beta$. A rule can be accompanied with some conditions specifying when the rule can be applied.

The *operator Ext_L* is specified by the corresponding forward rules given in Table 1. Given an L-normal model generator I, $Ext_L(I)$ is the least extension of I that contains all ground atoms in L-normal labeled form that are derivable from some atom of I using the rules specifying Ext_L. Note that $Ext_L(I)$ is an L-normal model generator if so is I.

Denote $Serial_L = \{\boxdot \langle \top \rangle_i \top \mid 1 \leq i \leq m$ and $\boxdot \langle \top \rangle_i$ is in L-normal form$\}$.

Let I be an L-normal model generator. The *standard L-model* of I is constructed by building an L-model for $Ext_L(I) \cup Serial_L$ according to the semantics of ground labeled modal operators, and formally is defined as follows. Let $W' = EdgeLabels^*$ (i.e. the set of finite sequences of elements of

$L = KD4I_g5_a$, L-MProlog		
$\preceq_L$ is defined in page 10.		
The L-normal form of modalities is defined in page 8.		
Rules specifying operators Ext_L, Sat_L, NF_L, rNF_L, $rSat_L$:		
(*Both sides of each rule are in almost L-normal labeled form.*)		
Ext_L	$\triangle\Box_i\alpha \to \triangle\Box_j\alpha$ if $g(i) \supset g(j)$	(1)
	$\triangle\Box_i\alpha \to \triangle\Box_i\Box_i\alpha$	(2)
Sat_L	the rules specifying Ext_L plus	
	$\triangle\langle F\rangle_i E \to \triangle\Box_i\Diamond_i E$ if $g(i)$ is a singleton	(3)
	$\triangle\nabla\nabla' E \to \triangle\Diamond_i E$ if $\Diamond_i \preceq_L \nabla$ and $\Diamond_i \preceq_L \nabla'$	(4)
NF_L	$\triangle\nabla_i\nabla'_i E \to \triangle\nabla'_i E$ if $g(i)$ is a singleton and ∇'_i is of the form $\Box_i$ or $\langle E\rangle_i$	(5)
rNF_L	$\triangle\nabla_i E \leftarrow \triangle\langle X\rangle_i\nabla_i E$ if $g(i)$ is a singleton, ∇_i is of the form $\Box_i$ or $\langle E\rangle_i$, and X is a fresh atom variable	(6)
$rSat_L$	$\triangle\Diamond_i E \leftarrow \triangle\langle X\rangle_i E$ for X being a fresh atom variable	(7)
	$\triangle\nabla_i\alpha \leftarrow \triangle\Box_j\alpha$ if $g(i) \subseteq g(j)$	(8)
	$\triangle\Diamond_i E \leftarrow \triangle\Diamond_j E$ if $g(i) \supset g(j)$	(9)
	$\triangle\Box_i\Box_i\alpha \leftarrow \triangle\Box_i\alpha$	(10)
	$\triangle\nabla_i\Diamond_i E \leftarrow \triangle\Diamond_i E$ if $g(i)$ is a singleton	(11)
	$\triangle\Diamond_i E \leftarrow \triangle\langle X\rangle_j\Diamond_i E$ if $g(i) \supseteq g(j)$ and X is a fresh atom variable	(12)

Table 1. A schema for semantics of $KD4I_g5_a$-MProlog

$\{\langle E\rangle_i \mid E \in \mathcal{B} \cup \{\top\}$ and $1 \leq i \leq m\})$, $\tau = \epsilon$, $H(\tau) = Ext_L(I) \cup Serial_L$. Let $R'_i \subseteq W' \times W'$ and $H(u)$, for $u \in W'$, $u \neq \tau$, be the least sets such that:

- if $\langle E\rangle_i\alpha \in H(w)$, then $R'_i(w, w\langle E\rangle_i)$ holds and $\{E, \alpha\} \subseteq H(w\langle E\rangle_i)$;
- if $\Box_i\alpha \in H(w)$ and $R'_i(w, w\langle E\rangle_i)$ holds, then $\alpha \in H(w\langle E\rangle_i)$.

Let R_i, for $1 \leq i \leq m$, be the least[4] extension of R'_i such that $\{R_i \mid 1 \leq i \leq m\}$ satisfies all the L-frame restrictions except seriality (which is cared by $Serial_L$). Let W be W' without worlds not accessible directly nor indirectly from τ via the accessibility relations R_i. We call the model graph $\langle W, \tau, R_1, \ldots, R_m, H\rangle$ the *standard L-model graph* of I, and its corresponding model M the *standard L-model* of I. $\{R'_i \mid 1 \leq i \leq m\}$ is called the *skeleton* of M. By the *standard* $\Diamond$-

[4] the least extension exists due to the assumption that all L-frame restrictions not concerning seriality are classical first-order Horn formulas

realization function on M we call the $\Diamond$-realization function σ defined as follows: if $R'_i(w, w\langle E\rangle_i)$ holds then $\sigma(w, \langle E\rangle_i) = w\langle E\rangle_i$, else $\sigma(w, \langle E\rangle_i)$ is undefined.

It can be shown that *the standard L-model of an L-normal model generator I is a least L-model of I.*

4.3 Fixpoint Semantics

We now consider the direct consequence operator $T_{L,P}$. Given an L-normal model generator I, how can $T_{L,P}(I)$ be defined? Based on the axioms of L, I is first extended to the *L-saturation* of I, denoted by $Sat_L(I)$, which is a set of atoms. Next, *L-instances of program clauses* of P are *applied* to the atoms of $Sat_L(I)$. This is done by the operator $T_{0L,P}$. The set $T_{0L,P}(Sat_L(I))$ is a model generator but not necessary in L-normal form. Finally, the *normalization operator NF_L* converts $T_{0L,P}(Sat_L(I))$ to an L-normal model generator. $T_{L,P}(I)$ is defined as $NF_L(T_{0L,P}(Sat_L(I)))$.

To compare modal operators we define $\preceq_L$ to be the least reflexive and transitive relation between modal operators such that $\Diamond_i \preceq_L \langle E\rangle_i \preceq_L \Box_i$, $\Diamond_i \preceq_L \langle X\rangle_i \preceq_L \Box_i$, and if $g(i) \subseteq g(j)$ then $\Box_i \preceq_L \Box_j$ and $\Diamond_j \preceq_L \Diamond_i$.

An atom $\nabla^{(1)} \dots \nabla^{(n)}\alpha$ is called an *L-instance* of an atom $\nabla^{(1')} \dots \nabla^{(n')}\alpha'$ if there exists a substitution θ such that $\alpha = \alpha'\theta$ and $\nabla^{(i)} \preceq_L \nabla^{(i')}\theta$ for all $1 \leq i \leq n$ (treating $\nabla^{(i')}$ as an expression). For example, if $g(1) \subseteq g(2)$ then $\Box_1\Diamond_2 E$ is an L-instance of $\Box_2\langle F\rangle_1 E$.

A modality $\triangle$ is called an *L-instance* of $\triangle'$, and we also say that $\triangle'$ is *equal to* or *more general in L than* $\triangle$ (hereby we define a *pre-order between modalities*), if $\triangle E$ is an L-instance of $\triangle' E$ for some ground classical atom E.

Let $\boxdot$ and $\boxdot'$ be universal modalities in L-normal form. We say that $\boxdot$ is an *L-context instance* of $\boxdot'$ if $\boxdot'\varphi \to \boxdot\varphi$ is L-valid (for every φ). (It can be shown that the propositional version of the logic L is decidable. So, the problem of checking whether a given universal modality is an L-context instance of another one is also decidable.)

Let $\boxdot$ and $\boxdot'$ be universal modalities in L-normal form, φ and φ' be program clauses with empty modal context. We say that $\boxdot\varphi$ is an *L-instance* of (a program clause) $\boxdot'\varphi'$ if $\boxdot$ is an L-context instance of $\boxdot'$ and there exists a substitution θ such that $\varphi = \varphi'\theta$.

For example, if $g(1) \subseteq g(2)$ then $\Box_2\Box_1$ is an L-context instance of $\Box_2$ and $\Box_2\Box_1(p(a) \leftarrow q(a))$ is an L-instance of $\Box_2(p(x) \leftarrow q(x))$.

We now give definitions concerning Sat_L, $T_{0L,P}$, and NF_L.

The *saturation operator Sat_L* is specified by the corresponding forward rules given in Table 1. Given an L-normal model generator I, $Sat_L(I)$ is the least extension of I that contains all ground atoms in almost L-normal labeled form that are derivable from some atom in I using the rules specifying Sat_L. For example, if $g(1)$ is a singleton and $g(2)$ is not, then $\Box_2\Box_2\Box_1\Diamond_1 p(a) \in Sat_L(\{\Box_2\langle q(b)\rangle_1\, p(a)\})$.

When computing the least fixpoint of a modal logic program, whenever an atom of the form $\triangle\Diamond_i E$ is introduced, we "fix" the $\Diamond$ by replacing the atom by

$\triangle\langle E\rangle_i E$. This leads to the following definition. The *forward labeled form* of an atom α is the atom α' such that if α is of the form $\triangle\Diamond_i E$ then $\alpha' = \triangle\langle E\rangle_i E$, else $\alpha' = \alpha$. For example, the forward labeled form of $\Diamond_1 s(a)$ is $\langle s(a)\rangle_1 s(a)$.

Let P be an L-MProlog program. The *operator* $T_{0L,P}$ is defined as follows: for a set I of ground atoms in almost L-normal labeled form, $T_{0L,P}(I)$ is the least (w.r.t. $\subseteq$) model generator such that if $\boxdot(A \leftarrow B_1, \ldots, B_n)$ is a ground L-instance of some program clause of P and $\triangle$ is a maximally general[5] ground modality in L-normal labeled form such that $\triangle$ is an L-instance of $\boxdot$ and $\triangle B_i$ is an L-instance of some atom of I (for every $1 \leq i \leq n$), then the forward labeled form of $\triangle A$ belongs to $T_{0L,P}(I)$.

For example, if $g(1) \subseteq g(2)$ and P contains the clause $\Box_2(\Diamond_1 p(x) \leftarrow q(x), r(x),\ \Box_1 s(x),\ \Diamond_2 t(x))$ and $I = \{\langle q(a)\rangle_1 q(a),\ \langle q(a)\rangle_1 r(a),\ \Box_2\Box_2 s(a),\ \Box_2\langle t(a)\rangle_1 t(a)\}$, then $\langle q(a)\rangle_1\langle p(a)\rangle_1 p(a) \in T_{0L,P}(I)$.

The *normalization operator* NF_L is specified by the corresponding forward rules given in Table 1. Given a model generator I, $NF_L(I)$ is the set of all ground atoms in L-normal labeled form that are derivable from some atom of I using the rules specifying NF_L. For example, if $g(1)$ is a singleton then $NF_L(\{\langle q(a)\rangle_1\langle p(a)\rangle_1 p(a)\}) = \{\langle p(a)\rangle_1 p(a)\}$.

Define $T_{L,P}(I) = NF_L(T_{0L,P}(Sat_L(I)))$. By definition, the operators Sat_L, $T_{0L,P}$, and NF_L are all increasingly monotonic and compact. Hence the operator $T_{L,P}$ is monotonic and continuous. By the Kleene theorem, it follows that $T_{L,P}$ has the least fixpoint $T_{L,P}\uparrow\omega = \bigcup_{n=0}^{\omega} T_{L,P}\uparrow n$, where $T_{L,P}\uparrow 0 = \emptyset$ and $T_{L,P}\uparrow n = T_{L,P}(T_{L,P}\uparrow(n-1))$ for $n > 0$. Denote the least fixpoint $T_{L,P}\uparrow\omega$ by $I_{L,P}$ and the standard L-model of $I_{L,P}$ by $M_{L,P}$.

It can be shown that for an L-MProlog program P, $M_{L,P}$ *is a least L-model of P*. See also Lemma 1 given in Section 5.

4.4 SLD-Resolution

The main work in developing an SLD-resolution calculus for L-MProlog is to specify a reverse analogue of the operator $T_{L,P}$. The operator $T_{L,P}$ is a composition of Sat_L, $T_{0L,P}$, and NF_L. So, we have to investigate reversion of these operators.

A *goal* is a clause of the form $\leftarrow \alpha_1, \ldots, \alpha_k$, where each α_i is an atom.

The following definition concerns reversion of the operator $T_{0L,P}$.

Let $G = \leftarrow \alpha_1, \ldots, \alpha_i, \ldots, \alpha_k$ be a goal and $\varphi = \boxdot(A \leftarrow B_1, \ldots, B_n)$ a program clause. Then G' is *derived* from G and φ in L using mgu θ, and called an *L-resolvent* of G and φ, if the following conditions hold:

- $\alpha_i = \triangle' A'$, with $\triangle'$ in L-normal labeled form, is called the *selected atom*, and A' is called the *selected head atom*;
- $\triangle'$ is an L-instance of a universal modality $\boxdot'$ and $\boxdot'(A \leftarrow B_1, \ldots, B_n)$ is an L-instance of the program clause φ;
- θ is an mgu of A' and the forward labeled form of A;

[5] w.r.t. the pre-order between modalities described earlier for L

– G' is the goal $\leftarrow (\alpha_1, \ldots, \alpha_{i-1}, \triangle' B_1, \ldots, \triangle' B_n, \alpha_{i+1}, \ldots, \alpha_k)\theta$.

For example, if $g(1) \subseteq g(2)$ then $\leftarrow \Box_1 \Diamond_2 q(x), \Box_1 r(x)$ is an L-resolvent of $\leftarrow \Box_1 p(x)$ and $\Box_2 (p(x) \leftarrow \Diamond_2 q(x), r(x))$ (here, $\boxdot = \Box_2$ and $\triangle' = \boxdot' = \Box_1$).

As a reverse analogue of the operator Sat_L, we provide the operator $rSat_L$, which is specified by the corresponding backward rules given in Table 1. We say that $\beta = rSat_L(\alpha)$ *using an* $rSat_L$ *rule* $\alpha' \leftarrow \beta'$ if $\alpha \leftarrow \beta$ is of the form $\alpha' \leftarrow \beta'$. We write $\beta = rSat_L(\alpha)$ to denote that "$\beta = rSat_L(\alpha)$ using some $rSat_L$ rule".

As a reverse analogue of the operator NF_L, we provide the operator rNF_L, which is specified by the corresponding backward rules given in Table 1. We say that $\beta =_\theta rNF_L(\alpha)$ *using an* rNF_L *rule* $\alpha' \leftarrow \beta'$ if θ is an mgu such that $\alpha\theta \leftarrow \beta$ is of the form $\alpha' \leftarrow \beta'$. We write $\beta =_\theta rNF_L(\alpha)$ to denote that "$\beta =_\theta rNF_L(\alpha)$ using some rNF_L rule". For example, if $g(1)$ is a singleton then we have $\langle Y \rangle_1 \langle E \rangle_1 E =_\theta rNF_L(\langle X \rangle_1 E)$ with $\theta = \{X/E\}$ and Y being a fresh atom variable.

Let $G = \leftarrow \alpha_1, \ldots, \alpha_i, \ldots, \alpha_k$ be a goal. If $\alpha'_i = rSat_L(\alpha_i)$ using an $rSat_L$ rule φ, then $G' = \leftarrow \alpha_1, \ldots, \alpha_{i-1}, \alpha'_i, \alpha_{i+1}, \ldots, \alpha_k$ is *derived* from G and φ, and we call G' an *(L-)resolvent* of G and φ, and α_i the *selected atom* of G.

Similarly, G' is *derived* from G and an rNF_L rule φ using an mgu θ, and called an *(L-)resolvent* of G and φ, if α_i is called the *selected atom*, $\alpha'_i =_\theta rNF_L(\alpha_i)$ using φ, and $G' = \leftarrow \alpha_1\theta, \ldots, \alpha_{i-1}\theta, \alpha'_i, \alpha_{i+1}\theta, \ldots, \alpha_k\theta$.

For example, resolving $\leftarrow \Box_1 \Box_1 p(x)$ with the rule $\triangle \Box_i \Box_i \alpha \leftarrow \triangle \Box_i \alpha$ results in $\leftarrow \Box_1 p(x)$, since $\triangle$ is instantiated to the empty modality, i is instantiated to 1, and α is instantiated to $p(x)$.

Observe that $rSat_L$ rules and rNF_L rules are similar to program clauses and the way of applying them is similar to the way of applying classical program clauses, except that we do not need mgu's for $rSat_L$ rules.

We now define SLD-derivation and SLD-refutation.

Let P be an L-MProlog program and G a goal. An *SLD-derivation* from $P \cup \{G\}$ in L consists of a (finite or infinite) sequence $G_0 = G, G_1, \ldots$ of goals, a sequence $\varphi_1, \varphi_2, \ldots$ of variants of program clauses of P, $rSat_L$ rules, or rNF_L rules, and a sequence $\theta_1, \theta_2, \ldots$ of mgu's such that if φ_i is a variant of a program clause or an rNF_L rule then G_i is derived from G_{i-1} and φ_i in L using θ_i, else $\theta_i = \varepsilon$ (the empty substitution) and G_i is derived from G_{i-1} and (the $rSat_L$ rule variant) φ_i. Each φ_i is called an *input clause/rule* of the derivation.

We assume *standardizing variables apart* as usual (see [16]).

An *SLD-refutation* of $P \cup \{G\}$ in L is a finite SLD-derivation from $P \cup \{G\}$ in L with the empty clause as the last goal in the derivation.

Let P be an L-MProlog program and G a goal. A *computed answer* θ in L of $P \cup \{G\}$ is the substitution obtained by restricting the composition $\theta_1 \ldots \theta_n$ to the variables of G, where $\theta_1, \ldots, \theta_n$ is the sequence of mgu's used in an SLD-refutation of $P \cup \{G\}$ in L.

4.5 Example

We give here an SLD-refutation of $P_{wise\_men} \cup \{\leftarrow \Box_a white(a)\}$ in $KD4I_g5_a$, where P_{wise_men} is the $KD4I_g5_a$-MProlog program given in Section 3.

Goals	Input clauses/rules	MGUs
$\leftarrow \Box_a\, white(a)$		
$\leftarrow \Box_a \Diamond_b\, white(a)$	φ_1	
$\leftarrow \Box_a \langle X_2 \rangle_b\, white(a)$	(7)	
$\leftarrow \Box_a \langle X_2 \rangle_b \Diamond_c\, white(a)$	φ_2	
$\leftarrow \Box_a \langle X_2 \rangle_b \langle X_4 \rangle_c\, white(a)$	(7)	
$\leftarrow \Box_a \langle X_2 \rangle_b \langle X_4 \rangle_c\, black(b), \Box_a \langle X_2 \rangle_b \langle X_4 \rangle_c\, black(c)$	φ_7	
$\leftarrow \Box_a \langle X_2 \rangle_b\, black(b), \Box_a \langle X_2 \rangle_b \langle X_4 \rangle_c\, black(c)$	φ_6	
$\leftarrow \Box_a \langle black(b) \rangle_b \langle X_4 \rangle_c\, black(c)$	φ_{10}	$\{X_2/black(b)\}$
$\diamond$	φ_{11}	$\{X_4/black(c)\}$

5 Soundness and Completeness

In this section, we prove soundness and completeness of the SLD-resolution calculus given for $KD4I_g5_a$-MProlog, which is stated as follows.

Theorem 1. *Let P be an $KD4I_g5_a$-MProlog program and G an $KD4I_g5_a$-MProlog goal. Then every computed answer in $KD4I_g5_a$ of $P \cup \{G\}$ is a correct answer in $KD4I_g5_a$ of $P \cup \{G\}$. Conversely, for every correct answer θ in $KD4I_g5_a$ of $P \cup \{G\}$, there exists a computed answer γ in $KD4I_g5_a$ of $P \cup \{G\}$ which is more general than θ (i.e. $\theta = \gamma\delta$ for some substitution δ).*

In [22], we presented a general framework for developing fixpoint semantics, the least model semantics, and SLD-resolution calculi for logic programs in multimodal logics, and proved that under certain expected properties of a concrete instantiation of the framework for a specific multimodal logic, the SLD-resolution calculus is sound and complete. The semantics of $KD4I_g5_a$-MProlog presented in the previous section and summarized in Table 1 is based on and compatible with the framework given in [22].

By the results of [22], to prove soundness and completeness of SLD-resolution of $KD4I_g5_a$-MProlog, we can prove Expected Lemmas 4 – 10 of [22] (w.r.t. the schema given in Table 1). The Expected Lemma 6 is trivial, and the Expected Lemmas 7 – 10, which concern properties of the operators Sat_L, NF_L, $rSat_L$, and rNF_L, can be verified in a straightforward way. The remaining Expected Lemmas 4 and 5 are renumbered respectively as Lemmas 1 and 2 given below.

A model generator I is called an *L-model generator of P* if $T_{L,P}(I) \subseteq I$.

Lemma 1. *Let P be an L-MProlog program and I an L-model generator of P. Then the standard L-model of I is an L-model of P.*

Lemma 2. *Let I be an L-normal model generator, M the standard L-model of I, and α a ground L-MProlog goal atom. Suppose that $M \models \alpha$. Then α is an L-instance of some atom of $Sat_L(I)$.*

To prove these lemmas we need Lemmas 3 and 4 given below.

If a modality $\triangle$ is obtainable from $\triangle'$ by replacing some (possibly zero) ∇_i by $\Box_i$ then we call $\triangle$ a *$\Box$-lifting form* of $\triangle'$. If $\triangle$ is a $\Box$-lifting form of $\triangle'$ then we call an atom $\triangle\alpha$ a $\Box$-*lifting form* of $\triangle'\alpha$. For example, $\Box_1\langle p(a)\rangle_1\Box_2 q(b)$ is a $\Box$-lifting form of $\langle X\rangle_1\langle p(a)\rangle_1\Diamond_2 q(b)$.

Lemma 3. *Let I be an L-normal model generator and $M = \langle W, \tau, R_1, \ldots, R_m, H\rangle$ the standard L-model graph of I. Let $w = \langle E_1\rangle_{i_1} \ldots \langle E_k\rangle_{i_k}$ be a world of M and $\triangle = w$ be a modality. Then for α not containing $\top$, $\alpha \in H(w)$ iff there exists a $\Box$-lifting form $\triangle'$ of $\triangle$ such that $\triangle'\alpha \in Ext_L(I)$.*

This lemma can be easily proved by induction on the length of $\triangle$.

The following lemma is labeled Expected Lemma 2 in [22]. It states that the standard L-model of I is really an L-model of I.

Lemma 4. *Let I be an L-normal model generator, M the standard L-model of I, and σ the standard $\Diamond$-realization function on M. Then M is an L-model and $M, \sigma \vDash I$.*

Proof. By the definition, M is an L-model. Let $\{R'_i \mid 1 \le i \le m\}$ be the skeleton of M. We prove by induction on the length of α that for any $w \in W$, if $\alpha \in H(w)$ then $M, \sigma, w \vDash \alpha$. The cases when α is a classical atom or $\alpha = \langle E\rangle_i\beta$ are trivial. Consider the remaining case when $\alpha = \Box_i\beta$. Let u be a world such that $R_i(w, u)$ holds. Because $Ext_L(I)$ contains only atoms in L-normal form and $\Box_i\beta \in H(w)$, there does not exist v such that $R'_i(v, w)$ holds. Consequently, since $R_i(w, u)$ holds, there exist worlds $w_0 = w$, w_1, …, w_{h-1}, $w_h = u$ and indices $j_1, \ldots, j_h$ with $h \ge 1$ such that $R'_{j_1}(w_0, w_1)$, …, $R'_{j_h}(w_{h-1}, w_h)$, and $g(k) \subseteq g(i)$ for all $k \in \{j_1, \ldots, j_h\}$. Since $\Box_i\beta \in H(w)$, by Lemma 3, there exists a $\Box$-lifting form $\triangle'$ of $\triangle = w$ such that $\triangle'\Box_i\beta \in Ext_L(I)$. By the rules specifying Ext_L, it follows that $\triangle'\Box_{j_1} \ldots \Box_{j_h}\beta \in Ext_L(I)$. Hence, by Lemma 3, $\beta \in H(u)$. By the inductive assumption, $M, \sigma, u \vDash \beta$. Hence $M, \sigma, w \vDash \Box_i\beta$.

Proof of Lemma 1 Let M be the standard L-model of I and σ the standard $\Diamond$-realization function on M. By the definition of L-instances of program clauses and the construction of M, it is sufficient to prove that for any ground L-instance $\boxdot(A \leftarrow B_1, \ldots, B_n)$ of some program clause of P, for any $w \in W$ being an L-instance of $\boxdot$, $M, w \vDash (A \leftarrow B_1, \ldots, B_n)$. Suppose that $M, w \vDash B_i$ for all $1 \le i \le n$. We show that $M, w \vDash A$.

Let $\triangle = w = \langle E_1\rangle_{i_1} \ldots \langle E_k\rangle_{i_k}$. We first show that for any ground simple atom B of the form E, $\Box_i E$, or $\Diamond_i E$, if $M, w \vDash B$ then $\triangle B$ is an L-instance of some atom from $Sat_L(I)$. Suppose that $M, w \vDash B$. If $k \ge 1$ and $i = i_k$ and $g(i)$ is a singleton, then let $v = \langle E_1\rangle_{i_1} \ldots \langle E_{k-1}\rangle_{i_{k-1}}$, else let $v = w$.

If $B = E$, then by Lemma 3, some $\Box$-lifting form of $\triangle B$ belongs to $Ext_L(I)$, and hence $\triangle B$ is an L-instance of some atom from $Sat_L(I)$.

Now suppose that $B = \Box_i E$. Let $u = v\langle\top\rangle_i$ and $\triangle' = v\Box_i$. We have $R_i(w, u)$, and hence $M, u \vDash E$. By Lemma 3, it follows that some $\Box$-lifting form of $\triangle' E$ belongs to $Ext_L(I)$. Hence, $\triangle B$ is an L-instance of some atom from $Sat_L(I)$.

Next, suppose that $B = \Diamond_i E$. Consider the case $w \ne v$ (i.e. $i = i_k$ and $g(i)$ is a singleton). Since $M, w \vDash B$, there exists F such that $v\langle F\rangle_i$ is a world of M and $M, v\langle F\rangle_i \vDash E$. Let $\triangle' = v\langle F\rangle_i$. By Lemma 3, some $\Box$-lifting form of $\triangle' E$ belongs to $Ext_L(I)$. Hence, by the rules (2) and (3) of Sat_L, $\triangle B$ is an L-instance of some atom from $Sat_L(I)$. Now consider the case $w = v$ (i.e. $k = 0$ or $i \neq i_k$ or $g(i)$ is not a singleton). Since $M, w \vDash \Diamond_i E$, there exists $u = w\langle F_1\rangle_{j_1} \dots \langle F_h\rangle_{j_h}$ such that $M, u \vDash E$, $h \geq 1$, and $g(l) \subseteq g(i)$ for all $l \in \{j_1, \dots, j_h\}$. By Lemma 3, some $\Box$-lifting form of $w\langle F_1\rangle_{j_1} \dots \langle F_h\rangle_{j_h} E$ belongs to $Ext_L(I)$. It follows that some $\Box$-lifting form of $\triangle\langle F_1\rangle_{j_1} \dots \langle F_h\rangle_{j_h} E$ belongs to $Sat_L(I)$. By the rules of Sat_L, some $\Box$-lifting form of $\triangle\Diamond_i E$ belongs to $Sat_L(I)$. Hence $\triangle B$ is an L-instance of some atom from $Sat_L(I)$.

Since $M, w \vDash B_i$ for $1 \leq i \leq n$, it follows that $\triangle B_i$ is an L-instance of some atom from $Sat_L(I)$. Consequently, $\triangle A$ is an L-instance of some atom α from $T_{0L,P}(Sat_L(I))$. Let α' be the L-normal form of α, i.e. $NF_L(\{\alpha\}) = \{\alpha'\}$. We have $\alpha' \in T_{L,P}(I) \subseteq I$. By Lemma 4, $M, \sigma \vDash \alpha'$. If $\alpha' = \alpha$ then we can derive from $M, \sigma \vDash \alpha'$ that $M, w \vDash A$. Suppose that $\alpha' \neq \alpha$. Thus, α is of the form $\triangle''\nabla_i\nabla'_i E$, where $\triangle''\nabla_i = \triangle$, $g(i)$ is a singleton, and ∇'_i is $\Box_i$ or $\langle E\rangle_i$. If $\nabla'_i = \langle E\rangle_i$ then $A = \Diamond_i E$. We have that $\alpha' = \triangle''\nabla'_i E$. Since $M, \sigma \vDash \alpha'$ and $g(i)$ is a singleton, it follows that $M, \sigma \vDash \triangle''\Box_i A$. Hence $M, w \vDash A$. This completes the proof.

Proof of Lemma 2 Let $\langle W, \tau, R_1, \dots, R_m, H\rangle$ be the standard L-model graph of I, $\boxdot = \Box_{i_1} \dots \Box_{i_k}$ be a modality, and $w = \langle\top\rangle_{i_1} \dots \langle\top\rangle_{i_k}$. Suppose that α is of the form $\boxdot E$. Since $M \vDash \alpha$, we have $M, w \vDash E$. Hence, by Lemma 3, $\boxdot E \in Ext_L(I)$, and we also have $\boxdot E \in Sat_L(I)$. Now suppose that α is of the form $\boxdot\Diamond_i E$ with the property that if $g(i)$ is a singleton then $i \neq i_k$. Since $M \vDash \alpha$, we have $M, w \vDash \Diamond_i E$. Hence there exists $u = w\langle F_1\rangle_{j_1} \dots \langle F_h\rangle_{j_h}$ such that $E \in H(u)$, $h \geq 1$, and $g(l) \subseteq g(i)$ for all $l \in \{j_1, \dots, j_h\}$. By Lemma 3, some $\Box$-lifting form of $w\langle F_1\rangle_{j_1} \dots \langle F_h\rangle_{j_h} E$ belongs to $Ext_L(I)$. It follows that some $\Box$-lifting form of $\boxdot\langle F_1\rangle_{j_1} \dots \langle F_h\rangle_{j_h} E$ belongs to $Ext_L(I)$ and $Sat_L(I)$. Hence $\boxdot\Diamond_i E$ is an L-instance of some atom from $Sat_L(I)$.

We have proved Lemmas 1 and 2, which completes the proof of Theorem 1.

6 Conclusions

Our contributions in this paper are: the schema for semantics of $KD4I_g5_a$-MProlog given in Table 1, proofs of the soundness and completeness of SLD-resolution for $KD4I_g5_a$-MProlog, and a formalization of the wise men puzzle in the purely logical formalism of $KD4I_g5_a$-MProlog together with its SLD-refutation.

In this text, we recalled a large number of definitions and constructions from [22] (which in turn is an extension of [19]) in order to make the paper self-contained and understandable. This does not reduce the originality of the above-mentioned contributions.

The SLD-refutation given in Section 4.5 for the wise men puzzle does not uses rules or properties involving with axiom (5_a). Consequently, the puzzle can be solved in the logic $KD4I_g = K_m + (D) + (4) + (I_g)$. The choice of $KD4I_g5_a$ is justified as one of possible multimodal logics of belief and common/mutual belief that can be used to formalize the wise men puzzle. Our framework for modal logic programming [22] is applicable for a wide class of multimodal logics and it can be extended for other Kripke model semantics (e.g. with varying domain or non-rigid terms).

This paper considers only one of different aspects of multi-agent systems. In particular, we did not consider temporal dimension, actions, and events. Thus the current version of MProlog is not yet an agent programming language like AgentSpeak(L) [25], 3APL [13], and KARO [18]. To deal with the mentioned aspects, possible solutions are to adopt CTL like the BDI-architecture [26], (concurrent) dynamic logic like the KARO system [18], or discrete linear temporal logic. Extending MProlog with dynamic logic or discrete linear temporal logic is possible, because such logics can be treated as modal logics. However, this is still not sufficient for practical multi-agent systems. There remain a lot of problems to be solved. In our opinion, multi-agent planning deserves for more attention. Also, perhaps we should use rewards and penalties for cooperative and competitive[6] multi-agent systems to deal with negotiation and cooperation. But in that case, it seems not easy to adopt logics for specification and verification of multi-agent systems.

In summary, this paper is on reasoning about common/mutual belief (which was also considered in the paper [18] on KARO, but neglected in [26, 25, 13]). It shows that the wise men puzzle can be nicely formalized in a multimodal logic of belief using modal logic programming. Our system is goal-driven and we focused on theoretical aspects like soundness and completeness.

References

1. H. Aldewereld, W. van der Hoek, and J.-J.Ch. Meyer. Rational teams: Logical aspects of multi-agent systems. *Fundamenta Informaticae*, 63(2–3):159–183, 2004.
2. G. Attardi and M. Simi. Proofs in context. In J. Doyle, E. Sandewall, and P. Torasso, editors, *KR'94: Principles of Knowledge Representation and Reasoning*, pages 16–26, San Francisco, 1994. Morgan Kaufmann.
3. Ph. Balbiani, L. Fariñas del Cerro, and A. Herzig. Declarative semantics for modal logic programs. In *Proceedings of the 1988 International Conference on Fifth Generation Computer Systems*, pages 507–514. ICOT, 1988.
4. M. Baldoni. Normal multimodal logics with interaction axioms. In D. Basin, M. D'Agostino, D.M. Gabbay, and L. Viganò, editors, *Labelled Deduction*, pages 33–57. Kluwer Academic Publishers, 2000.
5. M. Baldoni, L. Giordano, and A. Martelli. A framework for a modal logic programming. In *Joint International Conference and Symposium on Logic Programming*, pages 52–66. MIT Press, 1996.
6. A. Cimatti and L. Serafini. Multi-agent reasoning with belief contexts: The approach and a case study. In M. Wooldridge and N.R. Jennings, editors, *Proceedings of ECAI-94, LNCS 890*, pages 71–85. Springer, 1995.
7. N. de Carvalho Ferreira, M. Fisher, and W. van der Hoek. Practical reasoning for uncertain agents. In J.J. Alferes and J.A. Leite, editors, *Proceedings of JELIA'2004*, volume 3229 of *LNCS*, pages 82–94. Springer-Verlag, 2004.

[6] Environment can be treated as a competitive agent.

8. F. Debart, P. Enjalbert, and M. Lescot. Multimodal logic programming using equational and order-sorted logic. *Theoretical Comp. Science*, 105:141–166, 1992.
9. J.J. Elgot-Drapkin. Step-logic and the three-wise-men problem. In *AAAI*, pages 412–417, 1991.
10. R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi. *Reasoning About Knowledge.* MIT Press, 1995.
11. L. Fariñas del Cerro. Molog: A system that extends Prolog with modal logic. *New Generation Computing*, 4:35–50, 1986.
12. M. Fisher and R. Owens. An introduction to executable modal and temporal logics. In M. Fisher and R. Owens, editors, *Executable Modal and Temporal Logics, IJCAI'93 workshop*, pages 1–20. Springer, 1995.
13. K.V. Hindriks, F.S. De Boer, W. van der Hoek, and J.-J.Ch. Meyer. Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.
14. M. Kacprzak, A. Lomuscio, and W. Penczek. Bounded versus unbounded model checking for interpreted systems (invited talk at FAAMAS'03). In B. Dunin-Keplicz and R. Verbrugge, editors, *Proceedings of FAAMAS'03*, pages 5–20, 2003.
15. K. Konolige. Belief and incompleteness. Technical Report 319, SRI Inter., 1984.
16. J.W. Lloyd. *Foundations of Logic Programming, 2nd Ed.* Springer-Verlag, 1987.
17. J. McCarthy. First order theories of individual concepts and propositions. *Machine Intelligence*, 9:120–147, 1979.
18. J.-J.Ch. Meyer, F.S. de Boer, R.M. van Eijk, K.V. Hindriks, and W. van der Hoek. On programming KARO agents. *Logic Journal of the IGPL*, 9(2), 2001.
19. L.A. Nguyen. A fixpoint semantics and an SLD-resolution calculus for modal logic programs. *Fundamenta Informaticae*, 55(1):63–100, 2003.
20. L.A. Nguyen. Multimodal logic programming and its applications to modal deductive databases. Manuscript (served as a technical report), available on Internet at `http://www.mimuw.edu.pl/~nguyen/papers.html`, 2003.
21. L.A. Nguyen. The modal logic programming system MProlog. In J.J. Alferes and J.A. Leite, editors, *Proceedings of JELIA 2004, LNCS 3229*, pages 266–278. Springer, 2004.
22. L.A. Nguyen. The modal logic programming system MProlog: Theory, design, and implementation. Available at `http://www.mimuw.edu.pl/~nguyen/mprolog`, 2005.
23. A. Nonnengart. How to use modalities and sorts in Prolog. In C. MacNish, D. Pearce, and L.M. Pereira, editors, *Proceedings of JELIA'94, LNCS 838*, pages 365–378. Springer, 1994.
24. M.A. Orgun and W. Ma. An overview of temporal and modal logic programming. In D.M. Gabbay and H.J. Ohlbach, editors, *Proc. First Int. Conf. on Temporal Logic - LNAI 827*, pages 445–479. Springer-Verlag, 1994.
25. A.S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Proceedings of the 7th European Workshop MAAMAW*, volume 1038 of *LNCS*, pages 42–55. Springer, 1996.
26. A.S. Rao and M.P. Georgeff. Modeling rational agents within a BDI-architecture. In *KR*, pages 473–484, 1991.
27. R.A. Schmidt and D. Tishkovsky. Multi-agent logic of dynamic belief and knowledge. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *Proceedings of JELIA'2002*, volume 2424 of *LNAI*, pages 38–49. Springer, 2002.
28. W. van der Hoek and J.-J. Meyer. Modalities for reasoning about knowledge and uncertainties. In P. Doherty, editor, *Partiality, Modality, and Nonmonotonicity.* CSLI Publications, 1996.

Decision Procedure for a Fragment of Mutual Belief Logic with Quantified Agent Variables

Regimantas Pliuškevičius and Aida Pliuškevičienė

Institute of Mathematics and Informatics
Akademijos 4, Vilnius 08663, LITHUANIA
{regis, aida}@ktl.mii.lt

Abstract. A deduction-based decision procedure for a fragment of mutual belief logic with quantified agent variables ($MBQL$) is presented. A language of $MBQL$ contains variables and constants for agents. The language of $MBQL$ is convenient to describe properties of rational agents when the number of agents is not known in advance. The multi-modal logic $KD45_n$ extended with restricted occurrences of quantifiers for agent variables is a component of $MBQL$. For this logic loop-check-free sequent calculus is proposed. This calculus corresponds to contraction-free calculus and does not require to translate sequents in a certain normal form. Another new point of presented decision procedure is existentially invertible separation rules. For a sequent containing occurrences of mutual belief modality two type of loop-check can be used: for positive occurrences of mutual belief modality loop-check can be used to find non-logical (loop-type) axioms, and for negative ones — to establish a non-derivability criterion.

1 Introduction

Mutual belief (common knowledge) logics are multi-modal logics extended with mutual belief (common knowledge) and everybody believes (everybody knows) modalities. Sequent-like calculi (with analytic cut rule instead of loop-type axioms) and Hilbert-style calculi for propositional common knowledge logics (based on finite set of agents) are constructed in several works (see, e.g., [1], [4], [11]). In [6] Hilbert-style calculus for common knowledge logic with infinite set of agents is presented. This calculus involves some restrictions on cardinality of set of agents and contains rather complex axiom for everybody knows operator. Propositional Hilbert-type calculus for mutual belief logic (based on finite set of agents) is constructed in several works (see, e.g., [2]).

Propositional agent-based logics are often insufficient for more complex real world situations. First-order extensions of these logics are necessary whenever a cardinality of an application domain and/or the number of agents are not known in advance. In [14] it is described a rich logic $LORA$ (Logic of Rational Agents), based on a three-sorted first-order logic (containing variables for agents, actions and other individuals), multi-agent BDI logic, and a dynamic logic. In [10] a logic QLB (quantified logic for belief) with Barcan axiom containing variables for agents and other individuals is presented. The same idea as in [10] and [14], namely, *use of term as an agent*, is utilized in term-modal logics [5]. In [13] a decision procedure for a fragment of temporal logic of belief and actions with restricted occurrences of quantified agent and action variables is presented.

In this paper, a fragment of mutual belief logic with quantified agent variables ($MBQL$) is considered. Different from [5], [10] and [14], the language of $MBQL$ does not contain function symbols. The aim of this paper is to present a deduction-based decision procedure for $MBQL$. The presented decision procedure is based on sequent-like calculus MBQ with invertible rules (in some sense). Separation rules is an important point of presented decision procedure. These existentially invertible rules incorporate "bad" quantifier rules for agent variables, the rules for everybody believes modality, and rules for belief modalities. Some deduction tools similar to separation rules are used informally in [12] for propositional (single agent) BDI logic. A decision procedure for logic $KD45_n$ extended with restricted occurrences of quantifiers for agent variables is another important point. For this logic loop-check-free sequent calculus is proposed. This calculus corresponds to contraction-free sequent calculus. However, loop-check-free type sequent calculus *differs* from contraction-free sequent calculus. In contraction-free sequent calculus (see [3], [7]) duplication of the main formula in the premise of a rule is *eliminated at all*. In loop-check-free sequent calculus duplication of the main formula in the premise of a rule is *not eliminated* but applications of rules containing such duplications are *restricted*. It allows to eliminate loop-check and does not require to translate sequents in a certain normal form as in [7]. For a sequent containing occurrences of mutual belief modality two type of loop-check can be used: for positive occurrences of mutual belief modality loop-check can be used to find non-logical (loop-type) axioms, and for negative ones – to establish a non-derivability criterion.

Here a procedural approach of decidable logical calculi is used and we assume that the notions of a decidable calculus and a deduction-based decision procedure are identical.

The paper is organized as follows. In Section 2, the language and the semantics of the $MBQL$ are presented. In Section 3, auxiliary tools for the presented decision procedure are described. In Section 4, a decision algorithm is presented relying on the sequent calculus MBQ and some examples demonstrating the presented algorithm are given. In Section 5, a foundation of the decision algorithm is given.

2 Language and semantics of $MBQL$

The $MBQL$ consists of the multi-modal logic $KD45_n$ (doxastic logic or weak-$S5_n$) extended with restricted occurrences of quantifiers for agent variables and logic containing mutual belief and everybody believes modalities [2].

The *language* of $MBQL$ contains: (1) a set of propositional symbols P, $P_1, \ldots$, Q, $Q_1, \ldots$; (2) a set of agent constants $i, i_1, \ldots, a_1, \ldots, b_1, \ldots, (i, i_l, a_j, b_j \in \{1, \ldots\})$; (3) a set of agent variables $x, x_1, \ldots, y, y_1, \ldots$; (4) a set of belief modality of the shape $\mathbf{B}(t)$, where t is an agent term, i.e., an agent constant or an agent variable; everybody believes modality $\mathbf{EB}$; mutual belief modality $\mathbf{MB}$; (5) logical operators: $\supset$, $\wedge$, $\vee$, $\neg$, $\forall$, $\exists$.

Formula of $MBQL$ is defined inductively as follows: every propositional symbol is formula; if A, B are formulas, then $A \supset B$, $A \wedge B$, $A \vee B$, $\neg(A)$ are formulas; if i is an agent, A is a formula, then $\mathbf{B}(i)A$ is a formula; if x is an agent variable, A is a formula, $Q \in \{\forall, \exists\}$, then $Qx\,\mathbf{B}(x)A$ is a formula; if A is a formula, then $\mathbf{EB}(A)$ and $\mathbf{MB}(A)$ are formulas. The formula A is a *logical* one if A contains only logical operators and propositional symbols.

As it follows from definition of formula, we do not consider, for example, expressions of the shape $\forall x \exists y\,\mathbf{B}(x)\,\mathbf{B}(y)A$, but expressions of the shape $\forall x\,\mathbf{B}(x) \exists y\,\mathbf{B}(y)A$ are considered.

When the formula under consideration contains occurrences of operators $\mathbf{EB}$ and/or $\mathbf{MB}$ it is assumed that the number of agents is *finite*. In this case the formula $\forall x\,\mathbf{B}(x)A$ means informally the same as the formula $\bigwedge_{i=1}^{n} \mathbf{B}(i)A$ and the formula $\exists x\,\mathbf{B}(x)A$ – as the formula $\bigvee_{i=1}^{n} \mathbf{B}(i)A$. Since the exact number of agents is not known *in advance*, in general, we use formulas with quantified agent variables.

The formula $\mathbf{B}(i)A$ means "agent i believes A". Formal semantics of the formula $\mathbf{B}(i)A$ satisfies the semantics of the logic $KD45_n$. The formula $\mathbf{EB}(A)$ means "every agent believes A", i.e. $\mathbf{EB}(A) \equiv \bigwedge_{i=1}^{n} \mathbf{B}(i)A$. The formula $\mathbf{MB}(A)$ means: "A is mutual belief of all agents". Therefore we use only so-called *public* mutual belief modality and assume that there is *perfect communication* between agents. The formula $\mathbf{MB}(A)$ has the same meaning as the infinite formula $\bigwedge_{k \geq 1} \mathbf{EB}^k(A)$, where $\mathbf{EB}^1(A) = \mathbf{EB}(A)$, and $\mathbf{EB}^k(A) = \mathbf{EB}^{k-1}(\mathbf{EB}(A))$, if $k > 1$. Infinitary nature of the modality $\mathbf{MB}$ is explained in [14]. The modalities $\mathbf{MB}$ and $\mathbf{EB}$ behave as modality of logic $KD4$. In addition, these modalities satisfy an induction-like property:

$\mathbf{EB}(A) \wedge \mathbf{MB}(A \supset \mathbf{EB}(A)) \supset \mathbf{MB}(A)$.

All belief modalities can be nested. For example, formula $\mathbf{B}(i_1)\,\mathbf{B}(i_2)P$, where P is a proposition "John is a good programmer", means "agent i_1 believes that agent i_2 believes that John is a good programmer". The formula

$\exists x\,\mathbf{B}(x)\forall y\,\mathbf{B}(y)P$, where P means the same as above, means "some agent believes that each agent believes that John is a good programmer".

To define the formal semantics of the formula $Qx\,\mathbf{B}(x)A$ ($Q \in \{\forall, \exists\}$) we must present an interpretation of agent variables. Such interpretation is received by means of an assignment: $V \to D$ (agent assignment), where V is a set of agent variables, D is a domain of agent constants. A model M is a pair $< \mathcal{I}, \mathbf{a} >$, where $\mathbf{a}$ is an agent assignment, $\mathcal{I}$ is a tuple $< D, St, \pi, \mathbf{R} >$, where D is a domain of agent constants; St is a set of states; π is an interpretation function of the propositional variables; $\mathbf{R}$ is the accessibility relations. All these relations satisfy transitive, serial, and Euclidean properties.

The concept "formula A is valid in $M =< \mathcal{I}, \mathbf{a} >$ at the state $s \in St$" (in symbols $M, s \models A$) is defined by induction on the structure of the formula of $MBQL$. Let us define only the cases when A is $Qx\,\mathbf{B}(x)N$, where $Q \in \{\forall, \exists\}$ (other cases are defined analogously as in [2], [4], [11], [14]).

$M, s \models \forall x\,\mathbf{B}(x)N$ if and only if for every agent assignment $\mathbf{a}'$ which differs from $\mathbf{a}$ at most with respect to an agent constant i, $< \mathcal{I}, \mathbf{a}' > \models \mathbf{B}(i)N$;

$M, s \models \exists x\,\mathbf{B}(x)N$ if and only if for some agent assignment $\mathbf{a}'$ which differs from $\mathbf{a}$ at most with respect to an agent constant i, $< \mathcal{I}, \mathbf{a}' > \models \mathbf{B}(i)N$;

Along with formulas we consider *sequents*, i.e., formal expressions $A_1, \ldots, A_k \to B_1, \ldots, B_m$ where $A_1, \ldots, A_k$ ($B_1, \ldots, B_m$) is a multiset of formulas. The sequent is interpreted as the formula $\bigwedge_{i=1}^{k} A_i \supset \bigvee_{j=1}^{m} B_j$. A sequent S is a logical one if S contains only logical formulas.

Let us recall the notions of positive and negative occurrences.

A formula (or some symbol) occurs *positively* in some formula B if it appears within the scope of no negation sign or in the scope of an even number of the negation sign, once all the occurrences of $A \supset C$ have been replaced by $\neg A \vee C$; in the opposite case, the formula (symbol) occurs *negatively* in B. For a sequent $S = A_1, \ldots, A_k \to B_1, \ldots, B_m$ positive and negative occurrences are determined just like for the formula $\bigwedge_{i=1}^{k} A_i \supset \bigvee_{j=1}^{m} B_j$.

3 Some Auxiliary Tools of the Decision Algorithm

A presented decision procedure is based on a sequent calculus with invertible rules. All derivations are constructed as a backward derivations. In this section, we present the main auxiliary tools of the decision algorithm: logical calculus, reduction and separation rules, and contraction rules.

Let (j) be any rule of a sequent calculus. Rule (j) is applied to get the conclusion of (j) from the premises of (j). If rule (j) is backward applied, i.e., to get premises of (j) from the conclusion of (j) we have a "bottom-up application of (j)" instead of "application of (j)". The rule (j) is called *invertible* in a sequent calculus I, if the derivability in I of the conclusion of (j) implies the derivability in I of each premise of (j). If the rule (j) is invertible, the bottom-up application of (j) preserves the derivability.

A decidable *calculus Log* is defined by the axiom: $\Gamma, A \to \Delta, A$ (where A is the main formula of the axiom) and traditional invertible rules for logical operators $\supset, \vee, \wedge, \neg$.

A derivation in the calculus Log is constructed as a tree using the bottom-up applications of the rules. A derivation D is *successful* if each leaf of D is an axiom and *unsuccessful* if there exists a leaf which is not an axiom.

Let us define reduction rules by means of which a sequent is reduced to a set of sequents in some canonical forms (see below).

Reduction rules consist of the following rules:

– Logical rules: all the rules of the calculus Log and the following rules:

$$\frac{\Gamma \to \Delta, A[b/x]}{\Gamma \to \Delta, \forall\, xA}\,(\to\forall) \qquad \frac{A[b/x], \Gamma \to \Delta}{\exists\, xA, \Gamma \to \Delta}\,(\exists\to),$$

where the variable x is agent variable and agent constant b (called an eigen-constant) does not enter the conclusion of the rules.

– Rules for mutual belief:

$$\frac{\mathbf{EB}(A),\, \mathbf{EB}(\,\mathbf{MB}(A)), \Gamma \to \Delta}{\mathbf{MB}(A), \Gamma \to \Delta}\,(\,\mathbf{MB} \to)$$

$$\frac{\Gamma \to \Delta,\, \mathbf{EB}(A);\; \Gamma \to \Delta,\, \mathbf{EB}(\,\mathbf{MB}(A))}{\Gamma \to \Delta,\, \mathbf{MB}(A)}\,(\to\, \mathbf{MB}).$$

– Rule for everybody believes:

$$\frac{\Gamma \to \Delta, \bigwedge_{i=1}^{n} \mathbf{B}(i)A}{\Gamma \to \Delta,\, \mathbf{EB}(A)}\,(\to\, \mathbf{EB}),\ \text{ where } n \text{ is a number of agents.}$$

Remark 1 *We do not introduce reduction rule for everybody believes operator (corresponding to implication* $\mathbf{EB}(A) \supset \bigwedge_{i=1}^{n} \mathbf{B}(i)A$*, where n is a number of agents) because it is included in separation rules (see below).*

To define the separation rules some canonical forms of sequents are introduced. A sequent S is *a primary* sequent, if S is of the following shape:

$\Sigma_1, \forall\, \mathcal{B}\Gamma,\, \mathbf{EB}\Pi_1,\, \mathbf{MB}\Delta_1 \to \Sigma_2, \exists\, \mathcal{B}\Delta,\, \mathbf{EB}\Pi_2,\, \mathbf{MB}\Delta_2$, where

– for every i ($i \in \{1,2\}$) Σ_i is empty or consists of logical formulas;
– $\forall\, \mathcal{B}\Gamma$ denotes a list $\forall x\, \mathbf{B}(x)\Gamma_0,\, \mathbf{B}(1)\Gamma_1, \ldots,\, \mathbf{B}(n)\Gamma_n$, where $\forall x\, \mathbf{B}(x)\Gamma_0$ (denoted as $\mathbf{\Theta_1}$ below) is empty or consists of formulas of the shape $\forall x_j\, \mathbf{B}(x_j)M_j$, $j \in \{1, 2, \ldots\}$; $\mathbf{B}(l)\Gamma_l$, $1 \le l \le n$, is *empty* or consists of formulas of the shape $\mathbf{B}(l)C$;

– $\exists\,\mathcal{B}\Delta$ denotes a list $\exists x\,\mathbf{B}(x)\Delta_0,\ \mathbf{B}(1)\Delta_1,\ldots,\ \mathbf{B}(n)\Delta_n$, where $\exists x\,\mathbf{B}(x)\Delta_0$ is empty or consists of formulas of the shape $\exists x_j\,\mathbf{B}(x_j)N_j$, $j\in\{1,2,\ldots\}$; $\mathbf{B}(r)\Delta_r,\ 1\le r\le n$, is *empty* or consists of formulas of the shape $\mathbf{B}(r)D$;
– for every i ($i\in\{1,2\}$) $\mathbf{EB}\Pi_i$ ($\mathbf{MB}\Delta_i$) is empty or consists of formulas of the shape $\mathbf{EB}(A)$ ($\mathbf{MB}(A)$, correspondingly).

A sequent S is a *reduced primary* sequent, if S is a primary one not containing $\mathbf{MB}\Delta_i$ but $\Gamma,\Delta,\Pi_1,\Pi_2$ may contain modality $\mathbf{MB}$.

A reduced primary sequent S is an $\mathbf{EB}$-*pure reduced primary* one if S is of the following shape $\Sigma_1,\mathbf{\Theta_1},\ \mathcal{B}\tilde{\Gamma},\ \mathbf{EB}\Pi_1\to\Sigma_2,\ \mathbf{EB}\Pi_2$, where (1) $\mathbf{\Theta_1}=\forall x\,\mathbf{B}(x)\Gamma_0$; (2) $\mathcal{B}\tilde{\Gamma}$ is empty or denotes a list $\mathbf{B}(1)\Gamma_1,\ldots,\ \mathbf{B}(n)\Gamma_n$ such that n is a number of agents and for *every* l $(1\le l\le n)$ $\mathbf{B}(l)\Gamma_l$ *is not empty*; (3) at least one from $\mathbf{EB}\Pi_1,\ \mathbf{EB}\Pi_2$ is not empty. Otherwise, the sequent S is *non-*$\mathbf{EB}$*-pure reduced primary* one.

From the shape of the primary sequent it is easy to see that bottom-up applying logical rules each sequent can be reduced to a set of primary sequents. As it follows from the shape of reduced primary sequent, bottom-up applying all reduction rules each primary sequent can be reduced to a set of reduced primary sequents.

To avoid loop-check in considered extension of the logic $KD45_n$ let us introduce marks of two sorts and indices. The marks are used in separation rules for modalities $\mathbf{B}(t)$ and $\mathbf{EB}$. The *first sort mark* has the shape $\mathbf{\Upsilon}^*$ ($\mathbf{\Upsilon}^*\in\{\mathbf{B}^*(t),\ \mathbf{EB}^*,\ \mathbf{MB}^*\}$). The first sort mark is defined as follows: let a formula A is in the sphere of action of a marked modality $\mathbf{\Upsilon}^*$. Then an occurrence of any modality $\mathbf{\Upsilon}$ ($\mathbf{\Upsilon}\in\{\mathbf{B}(t),\ \mathbf{EB},\ \mathbf{MB}\}$) in A is marked by the first sort mark and $\mathbf{\Upsilon}^{**}=\mathbf{\Upsilon}^*$. Both positive and negative occurrences of modality $\mathbf{\Upsilon}$ may contain the first sort mark. The *second sort mark* has the shape $\mathbf{B}^-(t)$. Only positive occurrences of belief modality $\mathbf{B}(t)$ in a sequent may contain the second sort mark. This mark is essential to get loop-check-free derivations in considered extension of the logic $KD45_n$. Besides marked modalities we use *indexed formulas* of the shape $\exists x^\circ\,\mathbf{B}^k(x^\circ)A$, where $\exists x^\circ\in\{\varnothing,\exists x\}$ and $x^\circ=i$ if $\exists x^\circ=\varnothing$; an index k is empty or $k\in\{*^\circ 1,\ldots,*^\circ m\}$, where $*^\circ\in\{\varnothing,*\}$. Only *positive occurrences* of formulas of the shape $\exists x^\circ\,\mathbf{B}(x^\circ)A$ in the succedent of a sequent may contain the indices. In the index k of the shape $*^\circ l$ l denotes a number of bottom-up applications of a separation rule for belief modality with the same main formula.

Let us introduce separation rule for everybody believes modality $\mathbf{EB}$. The conclusion of this separation rule is a $\mathbf{EB}$-pure reduced primary sequent, such that logical part $\Sigma_1\to\Sigma_2$ is not derivable in the calculus Log.

Separation rule (SR_1) **for everybody believes modality** $\mathbf{EB}$:

$$\frac{\mathbf{\Theta_1^*}, \mathit{\Gamma}_0,\, \mathcal{B}^*\tilde{\mathit{\Gamma}},\, \tilde{\mathit{\Gamma}},\, \mathbf{EB}^*\mathit{\Pi}_1, \mathit{\Pi}_1 \to A^\circ}{\mathit{\Sigma}_1, \mathbf{\Theta_1},\, \mathcal{B}\tilde{\mathit{\Gamma}},\, \mathbf{EB}\mathit{\Pi}_1 \to \mathit{\Sigma}_2,\, \mathbf{EB}\mathit{\Pi}_2,\, \mathbf{EB}(A^\circ)}\,(SR_1),$$

where $\mathbf{\Theta_1}$ and $\mathcal{B}\tilde{\mathit{\Gamma}}$ are determined in the definition of $\mathbf{EB}$-pure reduced primary sequent; $\tilde{\mathit{\Gamma}}$ (obtained from $\mathcal{B}\tilde{\mathit{\Gamma}}$) denotes a list $\mathit{\Gamma}_1, \ldots, \mathit{\Gamma}_n$, where n is a number of agents; $\mathbf{EB}(A^\circ) \in \{\varnothing, \mathbf{EB}(A)\}$; if $\mathbf{EB}\mathit{\Pi}_2$, $\mathbf{EB}(A^\circ)$ is empty, then A° is empty, otherwise $A^\circ = A$.

The formula $\mathbf{EB}(A)$ in the rule (SR_1) is the *main formula* of this rule.

Let us introduce two separation rules for belief modality $\mathbf{B}(t)$ denoted as (SR_2) and (SR_3). The conclusion of these separation rules is a reduced primary sequent, such that logical part $\mathit{\Sigma}_1 \to \mathit{\Sigma}_2$ is not derivable in Log.

Separation rule (SR_2) **for belief modality** $\mathbf{B}(t)$:

$$\frac{\mathbf{\Theta_1^*}, \mathit{\Gamma}_0,\, \mathbf{B}^*(l)\mathit{\Gamma}_l, \mathit{\Gamma}_l,\, \mathbf{EB}^*\mathit{\Pi}_1, \mathit{\Pi}_1 \to \mathbf{\Theta_2},\, \mathbf{B}(r)\mathit{\Delta}_r, \exists x^\circ\, \mathbf{B}^\sigma(x^\circ)M, M}{\mathit{\Sigma}_1, \forall\,\mathcal{B}\mathit{\Gamma},\, \mathbf{EB}\mathit{\Pi}_1 \to \mathit{\Sigma}_2, \exists\,\mathcal{B}\mathit{\Delta}, \exists x^\circ\, \mathbf{B}^k(x^\circ)M,\, \mathbf{EB}\mathit{\Pi}_2}\,(SR_2),$$

where $\forall\,\mathcal{B}\mathit{\Gamma}$, $\exists\,\mathcal{B}\mathit{\Delta}$, and $\mathbf{\Theta_1}$ are determined in the definition of primary sequent; $\mathbf{\Theta_2}$ means $\exists x\, \mathbf{B}(x)\mathit{\Delta}_0$.

The formula $\exists x^\circ\, \mathbf{B}^k(x^\circ)M$ is the *main formula* of (SR_2); $\exists x^\circ \in \{\varnothing, \exists x\}$.

To define an index σ let us consider two cases.

(1) $\exists x^\circ = \varnothing$, then $x^\circ = i$ and $\exists x^\circ\, \mathbf{B}^k(x^\circ)M$ has a shape $\mathbf{B}^k(i)M$. In this case $l = r = i$, i.e., $\mathbf{B}(l)\mathit{\Gamma}_l$ and $\mathbf{B}(r)\mathit{\Delta}_r$ consist of formulas of the shape $\mathbf{B}(i)D$. The index σ is defined in the following way. Let ρ (η) be the number of negative (positive, correspondingly) occurrences of modalities $\mathbf{B}(i)$, $\mathbf{EB}$, $\mathbf{MB}$ in M; let $\tau_0, \tau_1, \ldots,\ \tau_n, \tau_{n+1}$ be the number of negative occurrences of modalities $\mathbf{B}(i)$, $\mathbf{EB}$, $\mathbf{MB}$ in $\mathit{\Gamma}_0, \mathit{\Gamma}_1, \ldots,\ \mathit{\Gamma}_n, \mathit{\Pi}_1$, respectively, and $\tau = \max(\tau_0, \tau_1, \ldots, \tau_n, \tau_{n+1})$, $\rho' = \max(\rho - \eta, \tau - \eta)$. Then $k \in \{*^\circ 0, \ldots, *^\circ \rho'\}$ (where $*^\circ \in \{\varnothing, *\}$), at the very beginning k is empty and is treated as $*^\circ 0$. The index σ is defined as follows: if $k = *^\circ l,\ l \in \{0, \ldots, \rho'\}$ and $l < \rho'$ then $\sigma = *^\circ(l+1)$; otherwise, i.e., if $k = *^\circ l$ and $l = \rho'$, then $\sigma = -$.

(2) $\exists x^\circ = \exists x$. In this case all pairs consisting from $\mathbf{B}(l)\mathit{\Gamma}_l$ ($1 \le l \le n$) and $\mathbf{B}(r)\mathit{\Delta}_r$ ($1 \le r \le n$) must be reset. The index σ is defined in the same way as in the case (1) replacing a modality $\mathbf{B}(i)$ with $\mathbf{B}(t)$, where t is any agent variable or any agent constant.

The separation rule (SR_2) corresponds to transitivity and Euclidean properties of belief modality.

Separation rule (SR_3) **for belief modality** $\mathbf{B}(t)$:

$$\frac{\mathbf{\Theta_1^*}, \mathit{\Gamma}_0,\, \mathbf{B}^*(l)\mathit{\Gamma}_l, \mathit{\Gamma}_l,\, \mathbf{EB}^*\mathit{\Pi}_1, \mathit{\Pi}_1 \to}{\mathit{\Sigma}_1, \forall\,\mathcal{B}\mathit{\Gamma},\, \mathbf{EB}\mathit{\Pi}_1 \to \mathit{\Sigma}_2, \exists\,\mathcal{B}\mathit{\Delta},\, \mathbf{EB}\mathit{\Pi}_2}\,(SR_3),$$

where $\forall \mathcal{B}\Gamma$, $\exists \mathcal{B}\Delta$, and $\mathbf{\Theta_1}$ are the same as in the rule (SR_2).

The rule (SR_3) corresponds to the serial property of belief modality.

During the reduction to primary and reduced primary sequents the following contraction rules are used.

Contraction rules. The rule allowing to replace A, A_1 with A (where A and A_1 coincide or are congruent ones [9]) is an ordinary contraction rule. The rules allowing to replace $\mathbf{B}^k(t)A$, $\mathbf{B}^\circ(t)A$, where $\circ \in \{\varnothing, *\}$, with $\mathbf{B}^k(t)A$, to replace $\mathbf{B}^k(t)A$, $\mathbf{B}^-(t)A$ with $\mathbf{B}^-(t)A$, and to replace $\mathbf{\Upsilon}^*A$, $\mathbf{\Upsilon}A$, where $\mathbf{\Upsilon} \in \{\mathbf{B}(t), \mathbf{EB}, \mathbf{MB}\}$, with $\mathbf{\Upsilon}^*A$ are marked contraction rules. Contraction rules are backward applied *implicitly* (together with other rules).

Some examples in next section demonstrate an application of the separation rules and the use of the marks/indexes.

4 Description of Decision Algorithm

In presented decision procedure for the extension of the logic $KD45_n$ loop-check-free sequent calculus is proposed. Such type calculi correspond to contraction-free calculus for modal logic. For a sequent containing different occurrences of mutual belief modality $\mathbf{MB}$ two kind of loop-check (saturation) are used: for positive occurrences of mutual belief modality loop-check is used to find non-logical (loop-type) axioms, and for negative ones loop-check (called degenerate saturation) is used to establish a non-derivability criterion.

So, along with the logical axioms, we use non-logical (loop-type) axioms (as in other works on temporal and agent-based logics with induction axioms, see, e.g., [12], [13]). First we define parametrically identical formulas and sequents. Namely, formulas A and A' are called parametrically identical ones (in symbols $A \approx A'$) if either $A = A'$, or A and A' are congruent [9], or differ only by the corresponding occurrences of eigen-constants of the rules $(\to \forall)$, $(\exists \to)$; moreover , the occurrences of modality $\mathbf{\Upsilon}$ and marked modality $\mathbf{\Upsilon}^*$, where $\mathbf{\Upsilon} \in \{\mathbf{B}(t), \mathbf{EB}, \mathbf{MB}\}$, are treated as coinciding. Sequents $S = A_1, \ldots, A_k \to A_{k+1}, \ldots, A_{k+m}$ and $S' = A'_1, \ldots, A'_k \to A'_{k+1}, \ldots, A'_{k+m}$ are parametrically identical (in symbols $S \approx S'$), if $\forall j$ $(1 \le j \le k+m)$ formulas A_j and A'_j are parametrically identical ones. We say that a sequent $S = \Gamma \to \Delta$ subsumes a sequent $S' = \Pi, \Gamma' \to \Delta', \Theta$ (in symbols $S \succeq S'$) if $\Gamma \to \Delta \approx \Gamma' \to \Delta'$ (in special case, $S = S'$). A sequent S' is *subsumed* by S.

To obtain a negative criterion of derivability for the extension of the logic $KD45_n$ let us introduce a notion of b-final sequent.

A primary sequent of the shape $\Sigma_1, \forall \mathcal{B}^*\Gamma, \mathbf{EB}^*\Pi_1, \mathbf{MB}^*\Delta_1 \to \Sigma_2, \exists \mathcal{B}^-\Delta$ (in special case, $\Sigma_2, \exists \mathcal{B}^-\Delta$ is empty), such that logical part of this sequent, namely, $\Sigma_1 \to \Sigma_2$ is not derivable in the calculus Log, is b-*final* sequent.

Let D be a derivation in some calculus and i be a branch in D. The primary sequent $S = \Gamma \rightarrow \Delta$ from the branch i is a *saturated* sequent if, in the branch i above S, there exists a subsumed by S primary sequent S', i.e., $S \succeq S'$.

Let $S = \Sigma_1, \forall \mathcal{B}\Gamma,\, \mathbf{EB}\Pi_1,\, \mathbf{MB}\Delta_1 \rightarrow \Sigma_2,\, \mathbf{EB}\Pi_2$ be a saturated primary sequent in a derivation D. Then S is *degenerated saturated* one if in D there exists a subsumed by S primary sequent S' of the shape $\Sigma'_1, \forall \mathcal{B}^* \Gamma',\, \mathbf{EB}^* \Pi'_1$, $\mathbf{MB}^* \Delta'_1 \rightarrow \Sigma'_2,\, \mathbf{EB}^\circ \Pi'_2$ ($\circ \in \{\varnothing, *\}$) such that (1) logical part of S' is not derivable in the calculus Log; (2) Π'_2 does not contain any positive occurrence of modality $\mathbf{MB}$.

A saturated primary sequent S is $\mathbf{MB}$- *saturated* if $S = \Gamma \rightarrow \Delta,\, \mathbf{MB}(A)$. Sequents subsumed by an $\mathbf{MB}$-saturated sequent will be used as non-logical axioms.

The decision algorithm for an arbitrary sequent is realized by means of a calculus for mutual belief (MBQ).

Calculus MBQ:

A calculus MBQ is obtained from the calculus Log adding the separation rules (SR_l) ($1 \leq l \leq 3$), the reduction rules, contraction rules, and non-logical axioms of the shape $\Gamma \rightarrow \Delta,\, \mathbf{MB}(A)$.

A derivation D in the calculus MBQ is an *ordered derivation*, if it consists of several levels and each level consists of bottom-up applications of reduction rules. In this derivation at each level, when a set consisting of only reduced primary sequents is received, all *possible* bottom-up applications of the separation rules to every reduced primary sequent are realized. Each bottom-up application of the separation rules provides a possibility to construct a *different* (in general) ordered derivation D_k ($k \geq 1$). Let in the level j it be possible to bottom-up apply the rule (SR_2) using as the main formula of this rule several formulas, namely, $\exists x_1^\circ B(x_1^\circ)M_1, \ldots, \exists x_r^\circ B(x_r^\circ)M_r$. In this case as the main formula of (SR_2) we choose a such formula $\exists x_i^\circ B(x_i^\circ)M_i$ which was previously used as the main formula of this rule in the level $j - k$ $(k \geq 1)$. A such tactic of construction of an ordered derivation is called *directed* one. To eliminate redundancy from constructed ordered derivation in each level we do not consider (for a while) a sequent which is subsumed by some sequent in the level.

The ordered derivation D_k is a successful one, if *each* leaf of D_k ends with axiom (either logical or non-logical). The notion of logical axiom is obvious. Let us consider the notion of non-logical axiom in more detail. Let in ordered derivation D there exists reduction of a primary sequent of the shape $S = \Gamma \rightarrow \Delta,\, \mathbf{MB}(A)$ to a set of primary sequents $S_1, \ldots, S_p$, where sequent S_k $(1 \leq k \leq p)$ has the shape $\Pi, \Gamma'_k \rightarrow \Theta, \Delta'_k,\, \mathbf{MB}(A')$ and is such that $\Gamma \rightarrow \Delta \approx \Gamma'_k \rightarrow \Delta'_k$ and $A \approx A'$. The sequent S belongs to i-th level of D and S_k belongs to $(i + l)$-th level of D $(l \geq 1)$. Then the sequents S_k are considered as non-logical ($\mathbf{MB}$-*loop-type*) axioms of MBQ. In Section 5 it will be justified that non-logical axioms are founded automatically and consist of some parts of an end sequent of D.

If there *exists* an ordered derivation D of sequent S such that in a leaf of *each* branch i of D there is either a logical axiom, or a non-logical axiom, then in both these cases $MBQ \vdash S$ (positive criterion of termination of the procedure). If in *all* possible ordered derivations D_k of a sequent S there *exists* a branch having a sequent which is either non-derivable in Log or degenerated saturated one or b-final one, then $MBQ \nvdash S$ (negative criterion of termination of the procedure).

In the next section it will be justify that for any sequent a process of construction of an ordered derivation always terminates and proceeds automatically.

Bottom-up application of the reduction rule $(\rightarrow \mathbf{MB})$ is *induction-free* one, if the left premise S' of this rule has a shape $\Gamma \rightarrow \Delta,\ \mathbf{EB}(A)$, where $\Delta,\ A$ do not contain positive occurrences of modality $\mathbf{MB}$. If $MBQ \vdash^{D} S'$ and D does not contain non-logical axioms then this bottom-up application is *successful*.

From the notion of an ordered derivation in MBQ we get the following

Lemma 1 (derivability criterion in MBQ) *Let S be an arbitrary sequent. Then $MBQ \vdash^{D} S$ if and only if each induction-free bottom-up application of the reduction rule $(\rightarrow \mathbf{MB})$ in D is successful.*

Let (SR_2^{+}) be the rule obtained from the rule (SR_2) changing a definition of the index σ. Namely, let $\exists x^{\circ}\, \mathbf{B}^{k}(x^{\circ})M$ be the main formula of the rule (SR_2^{+}), and $k \in \{*^{\circ}0, *^{\circ}1, \ldots\}$ (where $*^{\circ} \in \{\varnothing, *\}$), at the very beginning k is empty and is treated as $*^{\circ}0$; if $k = *^{\circ}l$ then $\sigma = *^{\circ}(l+1)$. Let a calculus MBQ^{+} is obtained from MBQ adding the rule (SR_2^{+}). An application of the rule (SR_2^{+}) in MBQ^{+} is degenerate if $\sigma \geq \rho' + 1$, where ρ' is determined in the same way as in the rule (SR_2).

Analogously as in [13] using induction on number of the degenerate application of the rule (SR_2^{+}) we can prove

Lemma 2 *If $MBQ^{+} \vdash S$ then $MBQ \vdash S$.*

From Lemma 2 and relying on directed tactic in construction of ordered derivation we get

Lemma 3 *Let D be an ordered derivation in MBQ. Let $\Sigma_1, \forall \mathcal{B}\Gamma,\ \mathbf{EB}\Pi_1 \rightarrow \Sigma_2, \exists \mathcal{B}\Delta, \exists x^{\circ}\, \mathbf{B}^{k}(x^{\circ})M,\ \mathbf{EB}\Pi_2$ be a conclusion of an application of the rule (SR_2) in D. Then the same positive occurrence of the formula $\exists x^{\circ}\, \mathbf{B}^{k}(x^{\circ})M$ may be the main formula of applications of the separation rule (SR_2) in D* **at most** *$\rho' + 1$ time, where ρ' is defined in the rule (SR_2).*

Let us demonstrate saturation-free ordered derivations in MBQ, i.e., all branches of constructed ordered derivations end with logical axioms.

Example 1 *(a) Let $S = \mathbf{B}(1)P \rightarrow \mathbf{B}(1)\,\mathbf{B}(1)(P \vee Q)$. We can bottom-up apply (SR_2) or (SR_3) to S. Bottom-up applying (SR_3) to S we get b-final sequent $\mathbf{B}^*(1)P, P \rightarrow$. Let us consider the possibility to bottom-up apply (SR_2) to S. For S we have $\rho = 0$, $\eta = 1$, $\tau = 0$, and $\rho' = 0$. Therefore bottom-up applying (SR_2) to S we get $\sigma = -$ and $S_1 = \mathbf{B}^*(1)P, P \rightarrow \mathbf{B}^-(1)\,\mathbf{B}(1)(P \vee Q)$, $\mathbf{B}(1)(P \vee Q)$. Again, we can bottom-up apply (SR_2) or (SR_3) to S_1. Let us apply (SR_2). We can bottom-up apply (SR_2) to S_1 only with $\mathbf{B}(1)(P \vee Q)$ as the main formula. Since for S_1 $\rho' = 0$, we get $\sigma = -$ and $S_2 = \mathbf{B}^*(1)P, P \rightarrow \mathbf{B}^-(1)\,\mathbf{B}(1)(P \vee Q)$, $\mathbf{B}^-(1)(P \vee Q), P \vee Q$. Bottom-up applying $(\rightarrow \vee)$ to S_2 we get an axiom. Therefore $MBQ \vdash S$.*

(b) Let $S = \mathbf{B}(1)\,\mathbf{EB}(P) \rightarrow \mathbf{B}(1)(P \vee Q)$, i.e., for S $\rho = 0, \eta = 0, \tau = 1$, and $\rho' = 1$. Therefore bottom-up applying (SR_2) to S we get $\sigma = 1$ and $S_1 = \mathbf{B}^*(1)\,\mathbf{EB}^*(P)$, $\mathbf{EB}(P) \rightarrow \mathbf{B}^1(1)(P \vee Q), (P \vee Q)$. Since for S_1 $\rho' = 1$ and $k = 1$, bottom-up applying (SR_2) to S_1 we get $S_2 = \mathbf{B}^*(1)\,\mathbf{EB}^*(P)$, $\mathbf{EB}^*(P)$, $P \rightarrow \mathbf{B}^-(1)(P \vee Q), P \vee Q$. Bottom-up applying $(\rightarrow \vee)$ to S_2 we get an axiom. Therefore $MBQ \vdash S$.

(c) Let $S = \mathbf{B}(1)\,\mathbf{EB}(\,\mathbf{B}(1)P) \rightarrow \mathbf{B}(1)A$, where $A = \neg\,\mathbf{B}(2)Q \vee P$. For S we have $\rho = 0, \eta = 0, \tau = 2$, and $\rho' = 2$. Therefore bottom-up applying (SR_2) to S we get $\sigma = 1$ and, after applying $(\rightarrow \vee)$, $(\rightarrow \neg)$, we get $S_1 = \mathbf{B}^*(1)\,\mathbf{EB}^*(\,\mathbf{B}^*(1)P)$, $\mathbf{EB}(\,\mathbf{B}(1)P)$, $\mathbf{B}(2)Q \rightarrow \mathbf{B}^1(1)A^*, P$. Since for S_1 $\rho' = 2$, bottom-up applying (SR_2) and $(\rightarrow \vee)$, $(\rightarrow \neg)$ from S_1 we get $\sigma = 2$ and $S_2 = \mathbf{B}^*(1)\,\mathbf{EB}^*(\,\mathbf{B}^*(1)P)$, $\mathbf{EB}^*(\,\mathbf{B}^*(1)P)$, $\mathbf{B}(1)P$, $\mathbf{B}(2)Q \rightarrow \mathbf{B}^2(1)A, P$. For S_2 we get again $\rho' = 2$. Bottom-up applying (SR_2), $(\rightarrow \vee)$, $(\rightarrow \neg)$ from S_2 we get $\sigma = -$ and $S_3 = \mathbf{B}^*(1)\,\mathbf{EB}^*(\,\mathbf{B}^*(1)P)$, $\mathbf{EB}^*(\,\mathbf{B}^*(1)$ $P)$, $\mathbf{B}^*(1)P, P$, $\mathbf{B}(2)Q \rightarrow \mathbf{B}^-(1)A, P$. Since S_3 is an axiom, $MBQ \vdash S$.

(d) Let $\{1, \ldots, n\}$ be a set of agent constants and $S = \mathbf{B}(1)P_1$, $\ldots\,\mathbf{B}(n)P_n \rightarrow \mathbf{EB}(\bigvee_{i=1}^{n} P_i)$. Bottom-up applying (SR_1) and then $(\rightarrow \vee)$ we get an axiom. Therefore $MBQ \vdash S$.

(e) Let $\{1, 2\}$ be a set of agent constants and $S = \mathbf{B}(1)P \rightarrow \mathbf{EB}(P \vee \neg\,\mathbf{B}(2)P)$. Bottom-up applying $(\rightarrow \mathbf{EB})$, $(\rightarrow \wedge)$ from S we get reduced primary sequents $S_1 = \mathbf{B}(1)P \rightarrow \mathbf{B}(1)(P \vee \neg\,\mathbf{B}(2)P)$ and $S_2 = \mathbf{B}(1)P \rightarrow \mathbf{B}(2)(P \vee \neg\,\mathbf{B}(2)P)$. Bottom-up applying (SR_2) and $(\rightarrow \vee)$ from S_1 we get an axiom. Bottom-up applying (SR_2) and $(\rightarrow \vee)$, $(\rightarrow \neg)$ from S_2 we get $S_3 = \mathbf{B}(2)P \rightarrow \mathbf{B}^1(2)(P \vee \neg\,\mathbf{B}(2)P), P$. Bottom-up applying (SR_2) and $(\rightarrow \vee)$ from S_3 we get an axiom. Therefore $MBQ \vdash S$.

Let us demonstrate negative criterion of termination, i.e., construction of ordered derivations in MBQ containing a branch which ends with b-final sequent or containing a degenerated saturated primary sequent.

Example 2 *(a) Let $S = \rightarrow \exists x\, \mathbf{B}(x)A$, where $A = \neg\, \mathbf{EB}(P) \vee Q$, i.e., for S $\rho = 1$ and $\rho' = 1$. Bottom-up applying (SR_2), and then $(\rightarrow \vee)$, $(\rightarrow \neg)$ from S we get $S_1 = \mathbf{EB}(P) \rightarrow \exists x\, \mathbf{B}^1(x)A, Q$. Since for S_1 $k = \rho' = 1$, bottom-up applying (SR_2), and then $(\rightarrow \vee)$, $(\rightarrow \neg)$ from S_1 we get $S_2 = \mathbf{EB}^*(P), P \rightarrow \exists x\, \mathbf{B}^-(x)A, Q$. S_2 is not an axiom and is b-final. Therefore $MBQ \nvdash S$.*

(b) Let $S = \mathbf{EB}(P), \mathbf{MB}(A) \rightarrow \mathbf{EB}(Q)$, where $A = P \supset \neg\, \mathbf{EB}(Q)$. Bottom-up applying $(\mathbf{MB} \rightarrow)$ to S we get $S_1 = \mathbf{EB}(P), \mathbf{EB}(\mathbf{MB}(A)), \mathbf{EB}(A) \rightarrow \mathbf{EB}(Q)$. Bottom-up applying (SR_1) to S_1 we get $S_2 = \mathbf{EB}^*(P), P, \mathbf{EB}^*(A), A, \mathbf{EB}^*(\mathbf{MB}^*(A)), \mathbf{MB}^*(A) \rightarrow Q$. Bottom-up applying $(\supset\rightarrow)$, $(\neg \rightarrow)$ from S_2 we get an axiom (with P as the main formula) and $S_3 = \mathbf{EB}^*(P), P, \mathbf{EB}^*(A), \mathbf{EB}^*(\mathbf{MB}^*(A)), \mathbf{MB}^*(A) \rightarrow \mathbf{EB}^*(Q), Q$. Since $S \succeq S_3$, from the shape of S_3 we get that S is a degenerated saturated sequent. Therefore $MBQ \nvdash S$.

Let S' be a sequent obtained from the sequent S replacing the formula A by $P \supset \neg\, \mathbf{B}(1)Q$. Then we get derivation ending with a b-final sequent.

Let us demonstrate a derivation in MBQ with MB-saturation, i.e., a constructed ordered derivation contains non-logical axioms along with logical ones.

Example 3 *Let S be $\mathbf{EB}(\forall x\, \mathbf{B}(x)P), \mathbf{MB}(A) \rightarrow \mathbf{MB}(\exists x\, \mathbf{B}(x)P)$, where $A = \exists x\, \mathbf{B}(x)P \supset \mathbf{EB}(\forall x\, \mathbf{B}(x)P)$. The sequent S is a modified version of induction axiom for modality $\mathbf{MB}$.*
Bottom-up applying $(\rightarrow \mathbf{MB})$ to S we get two sequents $S_1 = \mathbf{EB}(\forall x\, \mathbf{B}(x)P), \mathbf{MB}(A) \rightarrow \mathbf{EB}(\exists x\, \mathbf{B}(x)P)$ and $S_2 = \mathbf{EB}(\forall x\, \mathbf{B}(x)P), \mathbf{MB}(A) \rightarrow \mathbf{EB}(\mathbf{MB}(\exists x\, \mathbf{B}(x)P))$. Bottom-up applying $(\mathbf{MB} \rightarrow)$ to S_1 we get the sequent $S_1' = \mathbf{EB}(\forall x\, \mathbf{B}(x)P, \mathbf{EB}(A), \mathbf{EB}(\mathbf{MB}(A)) \rightarrow \mathbf{EB}(\exists x\, \mathbf{B}(x)P)$. Bottom-up applying (SR_1) to S_1' we get the sequent $S_1'' = \forall x\, \mathbf{B}(x)P, A, \mathbf{MB}(A), \Delta \rightarrow \exists x\, \mathbf{B}(x)P$, where $\Delta = \mathbf{EB}^*(\forall x\, \mathbf{B}^*(x)P), \mathbf{EB}^*(A), \mathbf{EB}^*(\mathbf{MB}^*(A))$. Bottom-up applying $(\supset\rightarrow)$ from S_1'' we get sequents $S_{11} = \forall x\, \mathbf{B}(x)P, \mathbf{MB}(A), \Delta \rightarrow \exists x\, \mathbf{B}(x)P$ and $S_{12} = \forall x\, \mathbf{B}(x)P, \mathbf{EB}(\forall x\, \mathbf{B}(x)P), \mathbf{MB}(A), \Delta \rightarrow \exists x\, \mathbf{B}(x)P$. Since $S_{11} \succeq S_{12}$, at first we consider the sequent S_{11}. Bottom-up applying $(\mathbf{MB} \rightarrow)$ from S_{11} we get $S_{11}' = \forall x\, \mathbf{B}(x)P, \Delta \rightarrow \exists x\, \mathbf{B}(x)P$. Bottom-up applying (SR_2) to S_{11}' we get an axiom with P as the main formula. Therefore $MBQ \vdash S_{11}$. Since $S_{11} \succeq S_{12}$, $MBQ \vdash S_{12}$ as well.

Now let us consider the sequent S_2. Bottom-up applying $(\mathbf{MB} \rightarrow)$ to S_2 we get $S_2' = \mathbf{EB}(\forall x\, \mathbf{B}(x)P), \mathbf{EB}(A), \mathbf{EB}(\mathbf{MB}(A)) \rightarrow \mathbf{EB}(\mathbf{MB}(\exists x\, \mathbf{B}(x)P))$. Bottom-up applying (SR_1) to S_2' we get the sequent $S_2'' = \forall x\, \mathbf{B}(x)P, A, \mathbf{MB}$

$(A), \Delta \to \mathbf{MB}(\exists x\, \mathbf{B}(x)P)$. *Bottom-up applying* $(\supset\to)$ *to* S''_2 *we get two sequents* $S_{21} = \forall x\, \mathbf{B}(x)P$, $\mathbf{MB}(A), \Delta \to \exists x\, \mathbf{B}(x)P$, $\mathbf{MB}(\exists x\, \mathbf{B}(x)P)$ *and* $S_{22} = \forall x\, \mathbf{B}(x)P$, $\mathbf{EB}(\forall x\, \mathbf{B}(x)P)$, $\mathbf{MB}(A), \Delta \to \mathbf{MB}(\exists x\, \mathbf{B}(x)P)$. *Since* $S \succeq S_{22}$, S *is* $\mathbf{MB}$*-saturated sequent. Now let us consider the sequent* S_{21}. *Bottom-up applying* $(\to \mathbf{MB})$ *and then* $(\mathbf{MB} \to)$, (SR_2) *in both branches of* $(\to \mathbf{MB})$ *we get an axiom with* P *as the main formula. Therefore* $MBQ \vdash S$.

5 Foundation of Presented Decision Procedure

To justify the presented decision procedure, we must found: (1) termination of the procedure, (2) invertibility of reduction and contraction rules, and existential invertibility (see below) of the separation rules in MBQ and (3) $\mathbf{MB}$-type saturated sequents as non-logical axioms. The termination will be founded by means of finiteness of the so-called R-subformulas of primary sequents which are generated during the construction of an ordered derivation. Let us define the notion of R-subformulas of a sequent.

Let S be a primary sequent and C be a formula entering S. A set of R-subformulas of C from S is denoted as $RSub(C)$ and defined inductively.

1. $RSub(P) = \varnothing$, where P is a logical formula.
2. $RSub(\mathbf{EB}(A)) = RSub(A)$.
3. $RSub(\neg A) = RSub(A)$.
4. $RSub(A \odot B) = \{RSub(A)\} \cup \{RSub(B)\}$, where $\odot \in \{\supset, \wedge, \vee\}$.
5. $RSub(\mathbf{B}(i)A) = \{\mathbf{B}(i)A\} \cup \{RSub(A)\}$.
6. $RSub(\mathbf{MB}(A)) = \{\mathbf{EB}(\mathbf{MB}(A))\} \cup \{RSub(\mathbf{EB}(A))\}$.
7. $RSub(Qx\, \mathbf{B}(x)A) = RSub(\mathbf{B}(c)A)$, where Q is $\forall(\exists)$ and Q occurs positively (negatively) in S, x is an agent variable and c is a new agent constant.
8. $RSub(Qx\, \mathbf{B}(x)A) = RSub(A)$, where Q is $\exists(\forall)$ and Q occurs positively (negatively) in S.

A set of R-subformulas of a sequent $S = A_1, \dots, A_k \to A_{k+1}, \dots, A_{k+m}$ is denoted by $RSub(S)$ and defined as $RSub(S) = \cup_{i=1}^{k+m} RSub(A_i)$. $R^*Sub(S)$ denotes a set obtained from $RSub(S)$ by merging parametrically identical formulas.

From definition of $R^*Sub(S)$ we get that the set $R^*Sub(S)$ is *finite*. This fact (along with Lemma 3) is crucial to obtain termination of presented procedure.

Analyzing the construction of an ordered derivation in MBQ we get that the presented decision procedure is exponential-time and $PSPACE$-complete, i.e., during the construction of an ordered derivation of a sequent S we generate the primary sequents the length of which can be restricted by some polynomial depending on $R^*Sub(S)$.

To justify the invertibility of reduction and contraction rules and the existential invertibility of the separation rules in MBQ an infinitary calculus MBQ_ω is introduced. MBQ_ω is obtained from MBQ by dropping the non-logical axiom, marks and indices in separation rules and replacing the reduction rule $(\to \mathbf{MB})$ by following infinitary rule:

$$\frac{\mathit{\Gamma} \to \mathit{\Delta},\, \mathbf{EB}(A); \ldots; \mathit{\Gamma} \to \mathit{\Delta},\, \mathbf{EB}^k(A); \ldots}{\mathit{\Gamma} \to \mathit{\Delta},\, \mathbf{MB}(A)}\ (\to \mathbf{MB}_\omega),$$

$k \in \{1, \ldots\}$; $\mathbf{EB}^1(A) = \mathbf{EB}(A)$, $\mathbf{EB}^k(A) = \mathbf{EB}(\mathbf{EB}^{k-1}(A))$, $k > 1$.

Using induction on $\mathcal{O}(D)$, where $\mathcal{O}(D)$ is the height of a derivation D [13] of the conclusion of a reduction rule in MBQ_ω, we can prove an invertibility of reduction rules (including $(\to \mathbf{MB}_\omega)$) in MBQ_ω.

Using reduction rules it is possible to construct a reduction of sequent S to a set $\{S_1, \ldots, S_m\}$, where S_j $(1 \leq j \leq m)$ is a primary (reduced primary) sequent automatically. Using the invertibility of reduction rules we get that if $MBQ_\omega \vdash S$ then $MBQ_\omega \vdash S_j$, $j \in \{1, \ldots, m\}$.

It is easy to see that the separation rules (SR_l) $(l \in \{1, 2, 3\})$ are not invertible in the usual way but they are existential invertible. The separation rule (SR_l) $(l \in \{1, 2, 3\})$ is *existential invertible* if from derivability of the conclusion of the separation rule (SR_l) follows that there exists at least one rule (SR_l) $(1 \leq l \leq 3)$ such that a premise of this rule is derivable. It is obvious that, in contrast to deterministic usual invertibility, existential invertibility is non-deterministic.

Using double induction on $< k(S), \mathcal{O}(D) >$, where $k(S)$ is a number of positive occurrences of modality $\mathbf{MB}$ in an end-sequent of the derivation D, we can prove an existential invertibility of separation rules.

Lemma 4 (existential invertibility of separation rules) *Let S be a reduced primary sequent, i.e., $S = \mathit{\Sigma}_1, \forall \mathcal{B}\mathit{\Gamma},\ \mathbf{EB}\mathit{\Pi}_1 \to \mathit{\Sigma}_2, \exists \mathcal{B}\mathit{\Delta},\ \mathbf{EB}\mathit{\Pi}_2$, such that $MBQ_\omega \vdash S$ and $Log \nvdash \mathit{\Sigma}_1 \to \mathit{\Sigma}_2$. Then either*

- *S is an $\mathbf{EB}$-pure reduced primary sequent and there exists a sequent S_1 such that $MBQ_\omega \vdash S_1 = \mathbf{\Theta}_\mathbf{1}^*, \mathit{\Gamma}_0,\ \mathcal{B}^*\tilde{\mathit{\Gamma}}, \tilde{\mathit{\Gamma}},\ \mathbf{EB}^*\mathit{\Pi}_1, \mathit{\Pi}_1 \to A^\circ$, where the sequent S_1 is defined in the formulation of the rule (SR_1), or*
- *there exists a formula $\exists x^\circ\, \mathbf{B}^k(x^\circ)M$ from $\exists \mathcal{B}\mathit{\Delta}$, such that $MBQ_\omega \vdash S_2 = \mathbf{\Theta}_\mathbf{1}^*, \mathit{\Gamma}_0,\ \mathbf{B}^*(l)\mathit{\Gamma}_l, \mathit{\Gamma}_l,\ \mathbf{EB}^*\mathit{\Pi}_1, \mathit{\Pi}_1 \ \to\ \mathbf{\Theta}_\mathbf{2},\ \mathbf{B}(r)\mathit{\Delta}_r, \exists x^\circ\, \mathbf{B}^\sigma(x^\circ)M, M$, where the sequent S_2 and the index σ is defined in the formulation of the rule (SR_2), or*
- *there exists $l \geq 0$ such that $MBQ_\omega \vdash S_3 = \mathbf{\Theta}_\mathbf{1}^*, \mathit{\Gamma}_0,\ \mathbf{B}(l)^*\mathit{\Gamma}_l, \mathit{\Gamma}_l,\ \mathbf{EB}^*\mathit{\Pi}_1, \mathit{\Pi}_1 \to$, where the sequent S_3 is defined in the formulation of the rule (SR_3).*

Using invertibility of the reduction and separation rules we can prove that the contraction rules are invertible in MBQ_ω

Using Schütte method (analogously as in [8]) we get

Theorem 1 (soundness and ω-completeness of MBQ_ω) *Let S be a sequent. Then $\forall M \models S \iff MBQ_\omega \vdash S$. The cut rule is admissible in MBQ_ω.*

From the fact that $MBQ_\omega \vdash \mathbf{MB}(A) \equiv \mathbf{EB}(A) \wedge \mathbf{EB}(\mathbf{MB}(A))$ and admissibility of cut in MBQ_ω we get that the rule $(\rightarrow \mathbf{MB})$ is admissible and invertible in MBQ_ω.

To get an equivalence between calculi MBQ and MBQ_ω we introduce invariant calculus $INMBQ$. To define this calculus let us introduce some auxiliary notions. Let $MBQ \vdash^D S$. Then a set of $\mathbf{MB}$-saturated sequents, i.e., the sequents of the shape $\Gamma \rightarrow \Delta, \mathbf{MB}(A)$, in D is denoted by $Sat\{S\}$. Let us decompose $Sat\{S\}$ into a set of sets $Sat^i\{S\}$ such that (1) $Sat\{S\} = \bigcup\limits_i Sat^i\{S\}$; (2) $\forall ij(Sat^i\{S\} \cap Sat^j\{S\}) = \varnothing$; (3) if $S_1, S_2 \in Sat^i\{S\}$, then S_1, S_2 have a common succedent member of the shape $\mathbf{MB}(A)$, which is called a *nucleus* of $Sat^i\{S\}$. Every set $Sat^i\{S\}$ is a component of decomposition of $Sat\{S\}$.

An *invariant calculus* $INMBQ$ is obtained from the calculus MBQ replacing the non-logical axioms by the following invariant rule:

$$\frac{\Gamma \rightarrow \Delta, I;\ I \rightarrow \mathbf{EB}(I);\ I \rightarrow \mathbf{EB}(A)}{\Gamma \rightarrow \Delta, \mathbf{MB}(A)}\ (\rightarrow \mathbf{MB}_I),$$

where the invariant formula I is constructed automatically.
The rule $(\rightarrow \mathbf{MB}_I)$ satisfies the following conditions:

- the conclusion of $(\rightarrow \mathbf{MB}_I)$, i.e., the sequent $S' = \Gamma \rightarrow \Delta, \mathbf{MB}(A)$ is such that $S' \in Sat^i\{S\}$ and $Sat^i\{S\}$ is $\{\Sigma_{i1} \rightarrow \Pi_{i1}, \mathbf{MB}(A); \ldots; \Sigma_{in} \rightarrow \Pi_{in}, \mathbf{MB}(A)\}$, where $\mathbf{MB}(A)$ is the nucleus of $Sat^i\{S\}$, i.e., S' is an $\mathbf{MB}$-saturated sequent from a derivation of a sequent S in MBQ;
- $I = \bigvee\limits_{j=1}^{n}((\exists\Sigma_{ij})^\wedge \wedge \neg(\forall\Pi_{ij})^\vee$; let Π be any set of formulas of the shape $\mathbf{B}(i_1)C_1, \ldots, \mathbf{B}(i_m)C_m$, where i_l $(1 \leq l \leq m))$ is an agent eigen-constant; then $Q\Pi = Qx_1\,\mathbf{B}(x_1)C_1, \ldots, Qx_m\,\mathbf{B}(x_m)C_m$, $Q \in \{\forall, \exists\}$ (therefore all the eigen-constants are correspondingly bounded); $\Gamma^\wedge(\Gamma^\vee)$ means the conjunction (disjunction, respectively) of formulas from Γ.

To prove that from $MBQ \vdash S$ follows $INMBQ \vdash S$, a derivation of each $\mathbf{MB}$-saturated sequent in $INMBQ$ must be constructed.

Example 4 *Let S be the same sequent as in Example 3, i.e., has the shape $\mathbf{EB}(\forall x\,\mathbf{B}(x)P), \mathbf{MB}(A) \rightarrow \mathbf{MB}(\exists x\,\mathbf{B}(x)P)$, where $A = \exists x\,\mathbf{B}(x)P \supset$*

$\mathbf{EB}(\forall x\, \mathbf{B}(x)P)$. *From Example 3 it follows that* S *is a* $\mathbf{MB}$*-saturated sequent. From definition of the invariant formula* I *we get* $I = \mathbf{EB}(\forall x\, \mathbf{B}(x)P) \wedge \mathbf{MB}(A)$. *It is easy to verify that* $Log \vdash \mathbf{EB}(\forall x\, \mathbf{B}(x)P), \mathbf{MB}(A) \rightarrow I$ (1)*;*

$INMBQ \vdash I \rightarrow \mathbf{EB}(I);$ (2) $INMBQ \vdash I \rightarrow \mathbf{EB}(A)$ (3).

Applying $(\rightarrow \mathbf{MB}_I)$ *to (1), (2) and (3) we get* $INMBQ \vdash S$.

Analogously as in [13] we get

$MBQ \vdash S \iff INMBQ \vdash S \iff MBQ_\omega \vdash S$ $(*)$.

From (*) we get that all reduction rules and contraction rules are invertible in MBQ and the separation rules are existentially invertible in MBQ.

From Theorem 1 and (*) follows that MBQ is *sound* and *complete*.

Using these facts, finiteness of $R^*Sub(S)$, and Lemma 3 we get the following

Theorem 2 *Let* S *be an arbitrary sequent. Then one can automatically construct a successful or unsuccessful ordered derivation* D *of the sequent* S *in* MBQ *such that* D *always terminates.*

References

1. Alberucci, L., The modal μ-calculus and the logic of common knowledge. *Ph.D. thesis, Institut für Informatic and angewandte mathematik, Universität Bern*, (2002).
2. Aldewereld, H., van der Hoek, W., Meyer, J.J.Ch., Rational teams: logical aspects of multi-agent systems. *Fundamenta Informaticae*, **63**(2-3), (2004), 159 – 183.
3. Dyckhoff, R., Contraction-free sequent calculi for intuitionistic logic, *Journal of Symbolic Logic*, **57**, (1992), 795–807.
4. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y., *Reasoning about knowledge*, MIT Press, Cambridge, Mass. (1995).
5. Fitting M., Thalmann L., Voronkov A., Term-modal logics. *Studia Logica*, **69**(1), (2001), 133-169.
6. Halpern, J.Y., Shore R.A., Reasoning about common knowledge with infinitely many agents. *Information and Computation*, **191** (2004), 1–40.
7. Hudelmaier, J., A contraction-free sequent calculus for $S4$, in *Proof Theory for Modal Logic*, H. Wansing, Ed. Kluwer Academic Publishers, Dordrechts, Boston/London (1996), 3–16.
8. Kawai, H., Sequential calculus for a first-order infinitary temporal logic. *Zeitchr. für Math. Logic and Grundlagen der Math.*, **33**, (1987), 423–432.
9. Kleene, S.C., *Introduction to metamathematics*, D.Van Nostrand Company, North-Holland Publishing Co., P. Noordhoff LTD (1952).
10. Lomuscio A., Colombetti M., QLB: A quantified logic for belief. *Lecture Notes in Artificial Intelligence*, **1193**, (1996), 71-85.
11. Meyer, J.J.Ch., van der Hoek, W., *Epistemic Logic for AI and Computer Science*. Cambridge University Press, Cambridge, 1995.
12. Nide, N., Takata, S., Deduction systems for BDI logics using sequent calculus. In *Proc. of AAMAS'02*, (2002), 928–935.
13. Pliuškevičius, R., Pliuškevičienė, A., Decision procedure for a fragment of temporal logic of belief and actions with quantified agent and action variables. *Annals of Mathematics, Computing* $\&$ *Teleinformatics*, **1**(2) (2004), 51–72.
14. Wooldridge, M., *Reasoning about Rational Agents*. The MIT Press (2000).

Strongly Complete Axiomatizations of “Knowing At Most” in Standard Syntactic Assignments

Thomas Ågotnes[1] and Michal Walicki[2]

[1] Department for Humanistic Informatics, University of Bergen
HF-bygget, Sydnesplass 7, 5007 Bergen, Norway
agotnes@ii.uib.no
[2] Department of Informatics, University of Bergen
PB. 7800, N-5020 Bergen, Norway
walicki@ii.uib.no

Abstract. Standard syntactic assignments (SSAs) model knowledge directly rather than as truth in all possible worlds as in modal epistemic logic, by assigning arbitrary truth values to atomic epistemic formulae. It is a very general approach to epistemic logic, but has no interesting logical properties — partly because the standard logical language is too weak to express properties of such structures. In this paper we extend the logical language with a new operator used to represent the proposition that an agent “knows at most” a given finite set of formulae and study the problem of strongly complete axiomatization of SSAs in this language. Since the logic is not semantically compact, a strongly complete *finitary* axiomatization is impossible. Instead we present, first, a strongly complete *infinitary* system, and, second, a strongly complete finitary system for a slightly weaker variant of the language.

1 Introduction

In traditional modal epistemic logic [1, 2], modelling knowledge as truth in all possible states in a Kripke structure, agents know all the logical consequences of their knowledge. It fails as a logic of the *computed* knowledge of real agents, because it assumes a very particular and extremely powerful mechanism for reasoning. In reality, different agents have different reasoning mechanisms (e.g. non-monotonic or resource-bounded) and representations of knowledge (e.g. as propositions or as syntactic formulae). Thus, a more general model of knowledge would be useful. A general approach is to model knowledge *directly* rather than as truth in all possible worlds. *Standard Syntactic Assignments (SSAs)* [1] is a syntactic approach in which a formula $K_i\phi$ is assigned a truth value independent of the truth value assigned to any other formula of the form $K_i\psi$. SSAs are generalizations of Kripke structures. In fact, it can be argued that SSAs is the *most* general model of knowledge. However, SSAs are so general that they have no interesting logical properties that can be expressed in the traditional language of epistemic logic – indeed, they are completely axiomatized by propositional logic.

In this paper, we extend the logical language with a new epistemic operator $\bigtriangledown_i$ for each agent. $\bigtriangledown_i X$, where X is a finite set of formulae, expresses the fact that agent i knows *at most* X. The main problem we consider is the construction of a strongly complete axiomatization of SSAs in this language. A consequence of the addition of the new operator is that semantic compactness is lost, and thus that a strongly complete finitary axiomatization is impossible. Instead we, first, present a strongly complete *infinitary* system, and, second, a strongly complete finitary system for SSAs for a slightly weaker variants of the epistemic operators. The results are a contribution to the logical foundation of multi-agent systems.

In Section 2 SSAs and their use in epistemic logic are introduced, before the "at most" operator $\bigtriangledown_i$ and its interpretation in SSAs is presented in Section 3. The completeness results are presented in Section 4, and we conclude and discuss related work in Section 5. We presently define some logical concepts and terminology used in the remainder.

1.1 Logic

By "a logic" we henceforth mean a language of formulae together with a satisfiability relation $\models$. The semantic structures considered in this paper each has a set of *states*, and satisfiability relations relate a formula to a pair consisting of a structure M and a state s of M. A formula ϕ is a (local) *logical consequence* of a theory (set of formulae) Γ, $\Gamma \models \phi$, iff $(M, s) \models \phi$ for all $\phi \in \Gamma$ implies that $(M, s) \models \phi$. The usual terminology and notation for Hilbert-style proof systems are used: $\Gamma \vdash_S \phi$ means that formula ϕ is derivable from theory Γ in system S, and when Δ is a set of formulae, $\Gamma \vdash_S \Delta$ means that $\Gamma \vdash_S \delta$ for each $\delta \in \Delta$. We use the following definition of maximality: a theory in a language L is maximal if it contains either ϕ or $\neg\phi$ for each $\phi \in L$. A logical system is *weakly complete*, or just *complete*, if $\models \phi$ (i.e. $\emptyset \models \phi$, ϕ is *valid*) implies $\vdash_S \phi$ (i.e. $\emptyset \vdash_S \phi$) for all formulae ϕ, and *strongly complete* if $\Gamma \models \phi$ implies $\Gamma \vdash_S \phi$ for all formulae ϕ and theories Γ. If a logic has a (strongly) complete logical system, we say that the logic *is* (strongly) complete. A logic is semantically *compact* if for every theory Γ, if every finite subset of Γ is satisfiable then Γ is satisfiable. It is easy to see that under the definitions used above:

Fact 1 A weakly complete logic has a sound and strongly complete finitary axiomatization iff it is compact.

2 The Epistemic Logic of Standard Syntactic Assignments

Standard Syntactic Assignments (SSAs) are defined and interpret the standard epistemic language, as follows. Given a number of agents n we write Σ for the set $\{1, \ldots, n\}$.

Definition 2 ($\mathcal{L}$) Given a set of primitive propositions Θ and a number of agents n, $\mathcal{L}(\Theta, n)$ (or just $\mathcal{L}$) is the least set such that:

– $\Theta \subseteq \mathcal{L}$
– If $\phi, \psi \in \mathcal{L}$ then $\neg\phi, (\phi \wedge \psi) \in \mathcal{L}$
– If $\phi \in \mathcal{L}$ and $i \in \Sigma$ then $K_i\phi \in \mathcal{L}$ □

The set of *epistemic atoms* is $\mathcal{L}^{At} = \{K_i\phi : \phi \in \mathcal{L}, i \in \Sigma\}$. An epistemic formula is a propositional combination of epistemic atoms. An SSA assigns a truth value to the primitive propositions and epistemic atoms.

Definition 3 (Standard Syntactic Assignment) A standard syntactic assignment (SSA) is a tuple

$$(S, \sigma)$$

where S is a set of states and

$$\sigma(s) : \Theta \cup \mathcal{L}^{At} \to \{\mathbf{true}, \mathbf{false}\}$$

for each $s \in S$. □

Satisfaction of an $\mathcal{L}$ formula ϕ by a state s of an SSA M, written $(M, s) \models \phi$, is defined as follows:

$$\begin{array}{lcl}
(M,s) \models p & \Leftrightarrow & \sigma(s)(p) = \mathbf{true} \\
(M,s) \models \neg\phi & \Leftrightarrow & (M,s) \not\models \phi \\
(M,s) \models (\phi \wedge \psi) & \Leftrightarrow & (M,s) \models \phi \text{ and } (M,s) \models \psi \\
(M,s) \models K_i\phi & \Leftrightarrow & \sigma(s)(K_i\phi) = \mathbf{true}
\end{array}$$

We note that although [1] define SSAs in a possible worlds framework, the question of satisfaction of ϕ in a state s does not depend on any other state $(((S,\sigma),s) \models \phi \Leftrightarrow ((\{s\},\sigma),s) \models \phi)$.

SSAs are very general descriptions of knowledge – in fact so general that no epistemic properties of the class of all SSAs can be described by the standard epistemic language:

Theorem 4 Propositional logic, with substitution instances for the language $\mathcal{L}$, is sound and complete with respect to SSAs. □

In the next section we increase the expressiveness of the epistemic language.

3 Knowing At Most

The formula $K_i\phi$ denotes that fact that i knows *at least* ϕ – he knows ϕ but he may know more. We can generalize this to finite sets X of formulae:

$$\triangle_i X \equiv \bigwedge\{K_i\phi : \phi \in X\}$$

representing the fact that i knows at least X. The new operator we introduce[3] in this paper is a dual to $\triangle_i$, denoting the fact that i knows *at most* X:

$$\triangledown_i X$$

[3] A similar operator is also used in [3] and [4].

denotes the fact that every formula an agent knows is included in X, but he may not know all the formulae in X. If $\mathcal{L}$ was finite, the operator $\triangledown_i$ could (like $\triangle_i$) be defined in terms of K_i:

$$\triangledown_i X = \bigwedge\{\neg K_i\phi : \phi \in \mathcal{L} \setminus X\}$$

But since $\mathcal{L}$ is not finite, $\triangledown_i$ is not definable by K_i. We also use a third, derived, epistemic operator: $\Diamond_i X \equiv \triangle_i X \wedge \triangledown_i X$ meaning that the agent knows exactly X. The extended language is called $\mathcal{L}_\triangledown$.

Definition 5 ($\mathcal{L}_\triangledown$) Given a set of primitive propositions Θ, and a number of agents n, $\mathcal{L}_\triangledown(\Theta, n)$ (or just $\mathcal{L}_\triangledown$) is the least set such that:

- $\Theta \subseteq \mathcal{L}_\triangledown$
- If $\phi, \psi \in \mathcal{L}_\triangledown$ then $\neg\phi, (\phi \wedge \psi) \in \mathcal{L}_\triangledown$
- If $\phi \in \mathcal{L}$ and $i \in \Sigma$ then $K_i\phi \in \mathcal{L}_\triangledown$
- If $X \in \wp^{fin}(\mathcal{L})$ and $i \in \Sigma$ then $\triangledown_i X \in \mathcal{L}_\triangledown$ $\square$

The language $\mathcal{L}_\triangledown(\Theta, n)$ is defined to express properties of SSAs over the language $\mathcal{L}(\Theta, n)$ (introduced in Section 2), and thus the epistemic operators K_i and $\triangledown_i$ operate on formulae from $\mathcal{L}(\Theta, n)$. We assume that Θ is countable, and will make use of the fact that it follows that $\mathcal{L}_\triangledown(\Theta, n)$ is (infinitely) countable.

If X is a finite set of $\mathcal{L}_\triangledown$ formulae, we write $\triangle_i X$ as a shorthand for $\bigwedge_{\phi\in X} K_i\phi$. In addition, we use $\Diamond_i X$ for $\triangle_i X \wedge \triangledown_i X$, and the usual derived propositional connectives.

The interpretation of $\mathcal{L}_\triangledown$ in a state s of an SSA M is defined in the same way as the interpretation of $\mathcal{L}$, with the following clause for the new epistemic operator:

$$(M, s) \models \triangledown_i X \quad \Leftrightarrow \quad \{\phi \in \mathcal{L} : \sigma(s)(K_i\phi) = \mathbf{true}\} \subseteq X$$

It is easy to see that

$$(M, s) \models \triangle_i X \quad \Leftrightarrow \quad \{\phi \in \mathcal{L} : \sigma(s)(K_i\phi) = \mathbf{true}\} \supseteq X$$
$$(M, s) \models \Diamond_i X \quad \Leftrightarrow \quad \{\phi \in \mathcal{L} : \sigma(s)(K_i\phi) = \mathbf{true}\} = X$$

3.1 Properties

The following schemata, where X, Y, Z range over finite sets of formulae and ϕ over single formulae, show some properties of SSAs, in the language $\mathcal{L}_\triangledown$.

$\triangle_i \emptyset$		E1
$\triangle_i X \wedge \triangle_i Y \rightarrow \triangle_i(X \cup Y)$		E2
$(\triangledown_i X \wedge \triangledown_i Y) \rightarrow \triangledown_i(X \cap Y)$		E3
$\neg(\triangle_i X \wedge \triangledown_i Y)$	when $X \not\subseteq Y$	E4
$(\triangledown_i(Y \cup \{\phi\}) \wedge \neg K_i\phi) \rightarrow \triangledown_i Y$		E5
$\triangle_i X \rightarrow \triangle_i Y$	when $Y \subseteq X$	**KS**
$\triangledown_i X \rightarrow \triangledown_i Y$	when $X \subseteq Y$	**KG**

It is straightforward to prove the following.

Lemma 6 E1–E5, **KG**, **KS** are valid. □

4 Axiomatizations of SSAs

In this section we discuss axiomatizations of standard syntactic assignments in the language $\mathcal{L}_\triangledown$. The following lemma shows that the logic is not compact, and thus it does not have a strongly complete finitary axiomatization (Fact 1).

Lemma 7 The logic of standard syntactic assignments in the language $\mathcal{L}_\triangledown$ is not compact. □

Proof Let $p \in \Theta$ and let Γ_1 be the following $\mathcal{L}_\triangledown$ theory:

$$\Gamma_1 = \{K_i p, \neg \triangledown_i \{p\}\} \cup \{\neg K_i \phi : \phi \neq p\}$$

Let Γ' be a finite subset of Γ_1. Clearly, there exists a ϕ' such that $\neg K_i \phi' \notin \Gamma'$. Let $M = (\{s\}, \sigma)$ be such that $\sigma(s)(K_i\phi) = \mathbf{true}$ iff $\phi = p$ or $\phi = \phi'$. It is easy to see that $(M, s) \models \Gamma'$. If there was some (M', s') such that $(M', s') \models \Gamma_1$, then $(M', s') \models \neg \triangledown_i \{p\}$ i.e. there must exist a $\phi \neq p$ such that $\sigma(s)(K_i\phi) = \mathbf{true}$ – which contradicts the fact that $(M', s') \models \neg K_i \phi$ for all $\phi \neq p$. Thus, every finite subset of Γ_1 is satisfiable, but Γ_1 is not. ■

We present a strongly complete *in*finitary axiomatization in 4.1. Then, in 4.2, a finitary axiomatization for a slightly weaker language than $\mathcal{L}_\triangledown$ is proven strongly complete for SSAs.

4.1 An Infinitary System

We define a proof system EC^ω for the language $\mathcal{L}_\triangledown$ by using properties presented in Section 3 as axioms, in addition to propositional logic. In addition, EC^ω contains an infinitary derivation clause **R***. After presenting EC^ω, the rest of the section is concerned with proving its strong completeness with respect to the class of all SSAs. This is done by the commonly used strategy of proving satisfiability of maximal consistent theories. Thus we need an infinitary variant of the Lindenbaum lemma. However, the usual proof of the Lindenbaum lemma for finitary systems is not necessarily applicable to infinitary systems. In order to prove the Lindenbaum lemma for EC^ω, we use the same strategy as [5] who prove strong completeness of an infinitary axiomatization of PDL (with canonical models). In particular, we use the same way of defining the derivability relation by using a weakening rule **W**, and we prove the deduction theorem in the same way by including a cut rule **Cut**.

Definition 8 ($\boldsymbol{EC^\omega}$) EC^ω is a logical system for the language $\mathcal{L}_\triangledown$ having the following axiom schemata

All substitution instances of tautologies of propositional calculus		**Prop**
$(\triangledown_i X \wedge \triangledown_i Y) \to \triangledown_i(X \cap Y)$		E3
$\neg(\triangle_i X \wedge \triangledown_i Y)$	when $X \not\subseteq Y$	E4
$(\triangledown_i(Y \cup \{\gamma\}) \wedge \neg K_i\gamma) \to \triangledown_i Y$		E5
$\triangledown_i X \to \triangledown_i Y$	when $X \subseteq Y$	**KG**

The derivation relation $\vdash_{EC^\omega}$ – written $\vdash_\omega$ for simplicity – between sets of $\mathcal{L}_\triangledown$ formulae and single $\mathcal{L}_\triangledown$ formulae is the smallest relation closed under the following conditions:

$\vdash_\omega \phi$ when ϕ is an axiom **Ax**

$\{\phi, \phi \to \psi\} \vdash_\omega \psi$ **MP**

$$\bigcup_{j\in J}\{\alpha_j \to \neg K_i\gamma : \gamma \notin X_j\} \vdash_\omega \bigwedge_{j\in J}\alpha_j \to \triangledown_i X \qquad \textbf{R*}$$

when $X = \bigcap_{j\in J} X_j$ and X and J are finite

$$\frac{\Gamma \vdash_\omega \phi}{\Gamma \cup \Delta \vdash_\omega \phi} \qquad \textbf{W}$$

$$\frac{\Gamma \vdash_\omega \Delta, \Gamma \cup \Delta \vdash_\omega \phi}{\Gamma \vdash_\omega \phi} \qquad \textbf{Cut}$$

In the above schemata, X, Y, Z, X_j range over sets of $\mathcal{L}$ formulae, γ over $\mathcal{L}$ formulae, Γ, Δ over sets of $\mathcal{L}_\triangledown$ formulae, ϕ, ψ, α_j over $\mathcal{L}_\triangledown$ formulae, and i over agents. J is some finite set of indices. □

It is easy to see that E1, E2 and **KS** are derivable in EC^ω.

The use of the weakening rule instead of more general schemas makes inductive proofs easier, but particular derivations can sometimes be more cumbersome. For example:

Lemma 9

$$\Gamma \cup \{\phi\} \vdash_\omega \phi \qquad \text{R1}$$

$$\frac{\vdash_\omega \psi \to \phi}{\Gamma \cup \{\psi \vdash_\omega \phi\}} \qquad \text{R2}$$

□

Proof

R1: $\{\phi, \phi \to \phi\} \vdash_\omega \phi$ by **MP**; $\vdash_\omega \phi \to \phi$ by **Ax**; $\{\phi\} \vdash_\omega \phi \to \phi$ by **W**; $\{\phi\} \vdash_\omega \phi$ by **Cut** and $\Gamma \cup \{\phi\} \vdash_\omega \phi$ by **W**.

R2: Let $\vdash_\omega \psi \to \phi$. By **W**, $\{\psi\} \vdash_\omega \psi \to \phi$; by **MP** $\{\psi, \psi \to \phi\} \vdash_\omega \phi$ and thus $\{\psi\} \vdash_\omega \phi$ by **Cut**. By **W**, $\Gamma \cup \{\psi\} \vdash_\omega \phi$. ■

In order to prove the Lindenbaum lemma, we need the deduction theorem. The latter is shown by first proving the following rule.

Lemma 10 The following rule of *conditionalization* is admissible in EC^ω.

$$\frac{\Gamma \cup \Delta \vdash_\omega \phi}{\Gamma \cup \{\psi \to \delta : \delta \in \Delta\} \vdash_\omega \psi \to \phi} \quad \textbf{Cond}$$

□

Proof The proof is by infinitary induction over the derivation $\Gamma \cup \Delta \vdash_\omega \phi$ (derivations are well-founded). The base cases are **Ax**, **MP** and **R***, and the inductive steps are **W** and **Cut**.

Ax: $\Gamma = \Delta = \emptyset$. We must show that $\vdash_\omega \psi \to \phi$ when $\vdash_\omega \phi$. By **W** we get $\phi \to (\psi \to \phi) \vdash_\omega \phi$, then $\phi, \phi \to (\psi \to \phi) \vdash_\omega \psi \to \phi$ is an instance of **MP**, and by **Cut** we get $\phi \to (\psi \to \phi) \vdash_\omega \psi \to \phi$. By **Prop**, $\vdash_\omega \phi \to (\psi \to \phi)$, so by **Cut** once more we get $\vdash_\omega \psi \to \phi$.

MP: $\Gamma \cup \Delta = \{\phi', \phi' \to \phi\} \vdash_\omega \phi$. That $\Gamma \cup \{\psi \to \delta : \delta \in \Delta\} \vdash_\omega \psi \to \phi$ can be shown for each of the four possible combinations of Γ and Δ in a similar way to the **Ax** case.

R*: $\phi = \wedge_{j \in J} \alpha_j \to \triangledown_i X$ and $\Gamma \cup \Delta = \cup_{j \in J}\{\alpha_j \to \neg K_i \phi' : \phi' \in \mathcal{L} \setminus X_j\}$ where J is finite and $X = \cap_{j \in J} X_j$ is finite, i.e. there exist for each $j \in J$ sets Y_j and Z_j such that $\mathcal{L} \setminus X_j = Y_j \uplus Z_j$ and

$$\Gamma = \bigcup_{j \in J} \{\alpha_j \to \neg K_i \phi' : \phi' \in Y_j\}$$
$$\Delta = \bigcup_{j \in J} \{\alpha_j \to \neg K_i \phi' : \phi' \in Z_j\}$$

Let

$$\Gamma' = \bigcup_{j \in J} \{(\psi \wedge \alpha_j) \to \neg K_i \phi' : \phi' \in Y_j\}$$
$$\Delta' = \bigcup_{j \in J} \{(\psi \wedge \alpha_j) \to \neg K_i \phi' : \phi' \in Z_j\}$$

$\Gamma' \cup \Delta' = \cup_{j \in J}\{(\psi \wedge \alpha_j) \to \neg K_i \phi' : \phi' \in \mathcal{L} \setminus X_j\}$, and thus $\Gamma' \cup \Delta' \vdash_\omega \gamma'$, where $\gamma' = \wedge_{j \in J}(\psi \wedge \alpha_j) \to \triangledown_i X$, by **R***. By **W**, $\Gamma' \cup \Delta' \cup \Gamma \vdash_\omega \gamma'$. By **Prop**, $\vdash_\omega (\alpha_j \to \neg K_i \phi') \to ((\psi \wedge \alpha_j) \to \neg K_i \phi')$ for each $\alpha_j \to \neg K_i \phi' \in \Gamma$, and by R2 (once for each formula in Γ) $\Delta' \cup \Gamma \vdash_\omega \Gamma'$. By **Cut**, $\Delta' \cup \Gamma \vdash_\omega \gamma'$, and it only remains to convert the conjunctions in Δ' and γ' to implications: $\Delta' \cup \Gamma \cup \{\gamma'\} \vdash_\omega \psi \to \phi$ by **Prop** and R2, and by **Cut** and **W** it follows that $\Delta' \cup \Gamma \cup \{\psi \to \delta : \delta \in \Delta\} \vdash_\omega \psi \to \phi$. By **Prop** and R2 (once of each formula in Δ), $\Gamma \cup \{\psi \to \delta : \delta \in \Delta\} \vdash_\omega \Delta'$, and by **Cut** $\Gamma \cup \{\psi \to \delta : \delta \in \Delta\} \vdash_\omega \psi \to \phi$, which is the desired conclusion.

W: $\Gamma' \cup \Delta' \vdash_\omega \phi$ for some $\Gamma' \subseteq \Gamma$ and $\Delta' \subseteq \Delta$. By the induction hypothesis we can use **Cond** to obtain $\Gamma' \cup \{\psi \to \delta : \delta \in \Delta'\} \vdash_\omega \psi \to \phi$, and thus $\Gamma \cup \{\psi \to \delta : \delta \in \Delta\} \vdash_\omega \psi \to \phi$ by **W**.

Cut: $\Gamma \cup \Delta \vdash_\omega \Delta'$ and $\Gamma \cup \Delta \cup \Delta' \vdash_\omega \phi$, for some Δ'. By the induction hypothesis on the first derivation (once for each $\delta' \in \Delta'$), $\Gamma \cup \{\psi \to \delta : \delta \in \Delta\} \vdash_\omega \phi \to \delta'$ for each $\delta' \in \Delta'$. By the induction hypothesis on the second derivation, $\Gamma \cup \{\psi \to \delta : \delta \in \Delta \cup \Delta'\} \vdash_\omega \psi \to \phi$. By **Cut**, $\Gamma \cup \{\psi \to \delta : \delta \in \Delta\} \vdash_\omega \psi \to \phi$. ∎

Theorem 11 (Deduction Theorem) The rule

$$\frac{\Gamma \cup \{\phi\} \vdash_\omega \psi}{\Gamma \vdash_\omega \phi \to \psi} \qquad \textbf{DT}$$

is admissible in EC^ω. □

Proof If $\Gamma \cup \{\phi\} \vdash_\omega \psi$, then $\Gamma \cup \{\phi \to \phi\} \vdash_\omega \phi \to \psi$ by **Cond**. $\Gamma \vdash_\omega \phi \to \phi$ by **Ax** and **W**, and thus $\Gamma \vdash_\omega \phi \to \psi$ by **Cut**. ∎

Now we are ready to show that consistent theories can be extended to maximal consistent theories. The proof relies on **DT**.

Lemma 12 (Lindenbaum lemma for EC^ω) If Γ is EC^ω-consistent, then there exists an $\mathcal{L}_\bigtriangledown$-maximal and EC^ω-consistent Γ' such that $\Gamma \subseteq \Gamma'$. □

Proof Recall **R***:

$$\bigcup_{j \in J} \{\alpha_j \to \neg K_k \psi : \psi \notin X_j\} \vdash_\omega \bigwedge_{j \in J} \alpha_j \to \bigtriangledown_k X.$$

Formulae which can appear on the right of $\vdash_\omega$ in its instances will be said to have **R***-form. A special case of this schema is when $\bigwedge_j \alpha_j$ is a tautology (i.e., each α_j is), from which

$$\bigcup_{j \in J} \{\neg K_k \phi : \psi \notin X_j\} \vdash_\omega \bigtriangledown_k X.$$

can be obtained. Now, $\Gamma' \supset \Gamma$ is constructed as follows. $\mathcal{L}_\bigtriangledown$ is countable, so let $\phi_1, \phi_2, \ldots$ be an enumeration of $\mathcal{L}_\bigtriangledown$ respecting the subformula relation (i.e., when ϕ_i is a subformula of ϕ_j then $i < j$).

$$\Gamma_0 = \Gamma$$

$$\Gamma_{i+1} = \begin{cases} \Gamma_i \cup \{\phi_{i+1}\} & \text{if } \Gamma_i \vdash_\omega \phi_{i+1} \\ \Gamma_i \cup \{\neg\phi_{i+1}\} & \text{if } \Gamma_i \not\vdash_\omega \phi_{i+1} \text{ and } \phi_{i+1} \text{ does not have the } \textbf{R*}\text{-form} \\ \Gamma_i \cup \{\neg\phi_{i+1}, K_k\psi\} & \text{if } \Gamma_i \not\vdash_\omega \phi_{i+1} \text{ and } \phi_{i+1} \text{ has the } \textbf{R*}\text{-form, where } \psi \text{ is} \\ & \text{arbitrary such that } \psi \notin X \text{ and } \Gamma_i \not\vdash_\omega \neg K_k\psi \end{cases}$$

$$\Gamma' = \bigcup_{i=0}^{\omega} \Gamma_i$$

The existence of ψ in the last clause in the definition of Γ_{i+1} is verified as follows: since $\Gamma_i \not\vdash_\omega \phi_{i+1}$, there must be, to prevent an application of **R***, at least one α_j and $\psi \notin X$ such that $\Gamma_i \not\vdash_\omega \alpha_j \to \neg K_k\psi$. By construction (and the ordering of formulae), each α_j or its negation is included in Γ_i. If $\Gamma_i \vdash_\omega \neg\alpha_j$ then also $\Gamma_i \vdash_\omega \alpha_j \to \neg K_k\psi$, and this would be the case also if $\Gamma_i \vdash_\omega \neg K_k\psi$. So $\Gamma_i \vdash_\omega \alpha_j$ and $\Gamma_i \not\vdash_\omega \neg K_k\psi$.

It is easy to see that Γ' is maximal.

We show that each Γ_i is consistent, by induction over i. For the base case, Γ_0 is consistent by assumption. For the inductive case, assume that Γ_i is consistent. Γ_{i+1} is constructed by one of the three cases in the definition:

1. Γ_{i+1} is obviously consistent.
2. If $\Gamma_{i+1} = \Gamma_i \cup \{\neg\phi_{i+1}\} \vdash_\omega \bot$, then $\Gamma_i \vdash_\omega \phi_{i+1}$ by **DT** and **Prop**, contradicting the assumption in this case.
3. Consider first the special case (when all α_j are tautologies). Assume that $\Gamma_{i+1} = \Gamma_i \cup \{\neg \triangledown_k X, K_k\psi\} \vdash_\omega \bot$. Then $\Gamma_i \vdash_\omega K_k\psi \to \triangledown_k X$ by **DT** and **Prop** and by E4, since $\psi \notin X$, $\Gamma_i \vdash_\omega K_k\psi \to \neg \triangledown_k X$, and thus $\Gamma_i \vdash_\omega \neg K_k\psi$ contradicting the assumption in this case.
   In the general case, assume that $\Gamma_{i+1} = \Gamma_i \cup \{\neg(\bigwedge_j \alpha_j \to \triangledown_k X), K_k\psi\} \vdash_\omega \bot$:
   i Then $\Gamma_i \vdash_\omega K_k\psi \to (\neg(\bigwedge_j \alpha_j \to \triangledown_k X) \to \bot)$, i.e., $\Gamma_i \vdash_\omega K_k\psi \to (\bigwedge_j \alpha_j \to \triangledown_k X)$, i.e., $\Gamma_i \vdash_\omega \bigwedge_j \alpha_j \to (K_k\psi \to \triangledown_k X)$.
   ii By assumption in the construction, $\Gamma_i \not\vdash_\omega \neg(\bigwedge_j \alpha_j)$ (for otherwise it would prove $\bigwedge_j \alpha_j \to \triangledown_k X$), but since $\bigwedge_j \alpha_j$ (as well as each α_j) is a subformula of ϕ_{i+1}, it or its negation is already included in Γ_i. But this means that $\Gamma_i \vdash_\omega \bigwedge_j \alpha_j$. Combined with (i), this gives $\Gamma_i \vdash_\omega K_k\psi \to \triangledown_k X$, i.e., $\Gamma_i \vdash_\omega \neg K_k\psi \vee \triangledown_k X$.
   iii On the other hand, by E4, since $\psi \notin X : \Gamma_i \vdash_\omega \neg(K_k\psi \wedge \triangledown_k X)$, i.e., $\Gamma_i \vdash_\omega \neg K_k\psi \vee \neg \triangledown_k X$. Combined with (ii) this means that $\Gamma_i \vdash_\omega \neg K_k\psi$, but this contradicts the assumption in the construction of Γ_{i+1}.

Thus each Γ_i is consistent.

To show that Γ' is consistent, we first show that

$$\Gamma'' \vdash_\omega \phi \Rightarrow (\Gamma'' \subseteq \Gamma' \Rightarrow \phi \in \Gamma') \tag{1}$$

holds for all derivations $\Gamma'' \vdash_\omega \phi$, by induction over the derivation. The base cases are **Ax**, **MP** and **R***, and the inductive steps are **W** and **Cut**. Let i be the index of the formula ϕ, i.e. $\phi = \phi_i$.

Ax: If $\vdash_\omega \phi$, then $\phi \in \Gamma_i$ by the first case in the definition of Γ_i.

MP: $\Gamma'' = \{\phi', \phi' \to \phi\}$. If $\Gamma'' \subseteq \Gamma'$, there exists k, l such that $\phi' \in \Gamma_k$ and $\phi' \to \phi \in \Gamma_l$. If $\phi \notin \Gamma'$, $\neg\phi \in \Gamma'$ by maximality, i.e. there exists a m such that $\neg\phi \in \Gamma_m$. But then $\neg\phi, \phi', \phi' \to \phi \in \Gamma_{\max(k,l,m)}$, contradicting consistency of $\Gamma_{\max(k,l,m)}$.

R*: $\Gamma'' = \cup_{j \in J}\{\alpha_j \to \neg K_k\psi : \psi \notin X_j\}$ and $\phi = \bigwedge_j \alpha_j \to \triangledown_k X$, where $X = \bigcap_j X_j$, and $\Gamma'' \subseteq \Gamma'$. If $\phi \notin \Gamma'$ then, by maximality, $\neg\phi \in \Gamma'$, and thus $\neg\phi \in \Gamma_i$. Then, by construction of Γ_i, $\Gamma_{i-1} \not\vdash_\omega \phi$ (otherwise $\phi \in \Gamma'$) and

$K_k\psi \in \Gamma_i$ for some $\psi \notin X$. By the same argument as in point 3.(ii) above, $\Gamma_i \vdash_\omega \bigwedge_j \alpha_j$, and hence also $\Gamma' \vdash_\omega \bigwedge_j \alpha_j$. But then, for an appropriate m (namely, for which $\phi_m = \alpha_j \rightarrow \neg K_k\psi$): $\Gamma_{m-1} \vdash_\omega \alpha_j$ and $\Gamma_{m-1} \vdash_\omega K_k\psi$, i.e., $\neg(\alpha_j \rightarrow \neg K_k\psi) \in \Gamma_m$, and so $\alpha_j \rightarrow \neg K_k\psi \notin \Gamma'$, which contradicts the assumption that $\Gamma'' \subseteq \Gamma'$.

W: $\Gamma'' = \Gamma''' \cup \Delta$, and $\Gamma''' \vdash_\omega \phi$. If $\Gamma'' \subseteq \Gamma'$, $\Gamma''' \subseteq \Gamma$ and by the induction hypothesis $\phi \in \Gamma'$.

Cut: $\Gamma'' \vdash_\omega \Delta$ and $\Gamma'' \cup \Delta \vdash_\omega \phi$. Let $\Gamma'' \subseteq \Gamma'$. By the induction hypothesis on the first derivation (once for each of the formulae in Δ), $\Delta \subseteq \Gamma'$. Then $\Gamma'' \cup \Delta \subseteq \Gamma'$, and by the induction hypothesis on the second derivation $\phi \in \Gamma'$.

Thus (1) holds for all $\Gamma'' \vdash_\omega \phi$; particularly for $\Gamma' \vdash_\omega \phi$. Consistency of Γ' follows: if $\Gamma' \vdash_\omega \bot$, then $\bot \in \Gamma'$, i.e. $\bot \in \Gamma_l$ for some l, contradicting the fact that each Γ_l is consistent. ■

The following Lemma is needed in the proof of the thereafter following Lemma stating satisfiability of maximal consistent theories.

Lemma 13 Let $\Gamma' \subseteq \mathcal{L}_\triangledown$ be an $\mathcal{L}_\triangledown$-maximal and EC^ω-consistent theory. If there exists an X' such that $\Gamma' \vdash_\omega \triangledown_i X'$, then there exists an X such that $\Gamma' \vdash_\omega \Diamond_i X$. □

Proof Let Γ' be maximal consistent, and let $\Gamma' \vdash_\omega \triangledown_i X'$. Let

$$X = \bigcap_{Y \subseteq X' \text{ and } \Gamma' \vdash_\omega \triangledown_i Y} Y$$

Since every Y is included in the finite set X', X is finite, and $\Gamma' \vdash_\omega \triangledown_i X$ can be obtained by a finite number of applications of E3. Let

$$Z = \bigcup_{\Gamma' \vdash_\omega \triangle_i Y} Y$$

If $\Gamma' \vdash_\omega \triangle_i Y$, then $Y \subseteq X$, since otherwise Γ' would be inconsistent by E4. Thus Z is finite. By a finite number of applications of E2, $\Gamma' \vdash_\omega \triangle_i Z$. If $Z \not\subseteq X$, then Γ' would be inconsistent by E4, so $Z \subseteq X$. We now show that $X \subseteq Z$. Assume the opposite: $\phi \in X$ but $\phi \notin Z$ for some ϕ. Let $X^- = X \setminus \{\phi\}$. $\Gamma' \not\vdash_\omega K_i\phi$, since otherwise $\phi \in Z$ by definition of Z. By maximality, $\Gamma' \vdash_\omega \neg K_i\phi$. By E5, $\Gamma' \vdash_\omega \triangledown_i X^-$ – but by construction of X it follows that $X \subseteq X^-$ which is a contradiction. Thus, $X = Z$, and $\Gamma' \vdash_\omega \Diamond_i X$. ■

Lemma 14 Every maximal EC^ω-consistent $\mathcal{L}_\triangledown$ theory is satisfiable. □

Proof Let Γ be maximal and consistent. We construct the following SSA, which is intended to satisfy Γ:

$$M^\Gamma = (\{s\}, \sigma^\Gamma)$$
$$\sigma^\Gamma(s)(p) = \textbf{true} \Leftrightarrow \Gamma \vdash_\omega p \text{ when } p \in \Theta$$
$$\sigma^\Gamma(s)(K_i\phi) = \textbf{true} \Leftrightarrow \phi \in X_i^\Gamma$$

where:

$$X_i^\Gamma = \begin{cases} Z \text{ where } \Gamma \vdash_\omega \diamondsuit_i Z & \text{if there is an } X' \text{ such that } \Gamma \vdash_\omega \triangledown_i X' \\ \{\gamma : \Gamma \vdash_\omega K_i\gamma\} & \text{otherwise} \end{cases}$$

In the definition of X_i^Γ, the existence of a Z such that that $\Gamma \vdash_\omega \diamondsuit_i Z$ in the case that there exists an X' such that $\Gamma \vdash_\omega \triangledown_i X'$ is guaranteed by Lemma 13. We show, by structural induction over ϕ, that

$$(M^\Gamma, s) \models \phi \iff \Gamma \vdash_\omega \phi \tag{2}$$

This is a stronger statement than the lemma; the lemma is given by the direction to the left. We use three base cases: when ϕ is in Θ, $\phi = K_i\psi$ and $\phi = \triangledown_i X$. The first base case and the two inductive steps negation and conjunction are trivial, so we show only the two interesting base cases. For each base case we consider the situations when X_i^Γ is given by a) the first and b) the second case in its definition.

- $\phi = K_i\psi$: $(M^\Gamma, s) \models K_i\psi$ iff $\psi \in X_i^\Gamma$.
  $\Rightarrow$) Let $\psi \in X_i^\Gamma$. In case a), $X_i^\Gamma = Z$ where $\Gamma \vdash_\omega \diamondsuit_i Z$ and by **KS**, $\Gamma \vdash_\omega K_i\psi$. In case b), $\Gamma \vdash_\omega K_i\psi$ by construction of X_i^Γ.
  $\Leftarrow$) Let $\Gamma \vdash_\omega K_i\psi$. In case a), $\Gamma \vdash_\omega \triangledown_i Z$ and thus $\psi \in Z = X_i^\Gamma$ by E4 and consistency of Γ. In case b), $\psi \in X_i^\Gamma$ by construction.
- $\phi = \triangledown_i X$: $(M^\Gamma, s) \models \triangledown_i X$ iff $X_i^\Gamma \subseteq X$.
  $\Rightarrow$) Let $X_i^\Gamma \subseteq X$. In case a), $\Gamma \vdash_\omega \diamondsuit_i Z$ where $Z = X_i^\Gamma \subseteq X$, so $\Gamma \vdash_\omega \triangledown_i X$ by **KG**. In case b), X_i^Γ must be finite, since X is finite. For any $\psi \notin X_i^\Gamma$, $\Gamma \nvdash_\omega K_i\psi$ by construction of X_i^Γ, and $\Gamma \vdash_\omega \neg K_i\psi$ by maximality. Thus, by **R*** (with $J = \{1\}$, $\alpha_1 = \top$ and $X_1 = X_i^\Gamma$), $\Gamma \vdash_\omega \triangledown_i X_i^\Gamma$, contradicting the assumption in case b). Thus, case b) is impossible.
  $\Leftarrow$) Let $\Gamma \vdash_\omega \triangledown_i X$. In case a), $\Gamma \vdash_\omega \triangle_i Z$ and by E4 and consistency of Γ $X_i^\Gamma = Z \subseteq X$. Case b) is impossible by definition. ■

Theorem 15 *EC^ω* is a sound and strongly complete axiomatization of standard syntactic assignments, in the language $\mathcal{L}_\triangledown$. □

Proof Soundness follows from Lemma 6, and the easily seen facts that **MP** and **R*** are logical consequences and that **W** and **Cut** preserve logical consequence, by induction over the definition of the derivation relation. Strong completeness follows from Lemmas 12 and 14. ■

4.2 A System for a Weaker Language

In the previous section we proved strong completeness of EC^ω by using **R***. It turns out that strong completeness can be proved without **R***, if we restrict the logical language slightly. The restriction is that for some arbitrary primitive proposition $\hat{p} \in \Theta$, $K_i\hat{p}$ and $\triangledown_i X$ are not well-formed formulae for any i and any X with $\hat{p} \in X$. The semantics is not changed; we are still interpreting the language in SSAs over $\mathcal{L}(\Theta, n)$ as described in Sections 2 and 3. Thus, in the restricted logic agents can know something which is not expressible in the logical language.

$\mathcal{L}^{\hat{p}}_{\bigtriangledown} \subset \mathcal{L}_{\bigtriangledown}$ is the restricted language for a given primitive proposition $\hat{p}$.

Definition 16 ($\mathcal{L}^{\hat{p}}_{\bigtriangledown}$) Given a set of primitive propositions Θ, a proposition $\hat{p} \in \Theta$ and a number of agents n, $\mathcal{L}^{\hat{p}}_{\bigtriangledown}(\Theta, n)$ (or just $\mathcal{L}^{\hat{p}}_{\bigtriangledown}$) is the least set such that:

- $\Theta \subseteq \mathcal{L}^{\hat{p}}_{\bigtriangledown}$
- If $\phi, \psi \in \mathcal{L}^{\hat{p}}_{\bigtriangledown}$ then $\neg\phi, (\phi \wedge \psi) \in \mathcal{L}^{\hat{p}}_{\bigtriangledown}$
- If $\phi \in (\mathcal{L} \setminus \{\hat{p}\})$ and $i \in \Sigma$ then $K_i\phi \in \mathcal{L}^{\hat{p}}_{\bigtriangledown}$
- If $X \in \wp^{fin}(\mathcal{L} \setminus \hat{p})$ and $i \in \Sigma$ then $\bigtriangledown_i X \in \mathcal{L}^{\hat{p}}_{\bigtriangledown}$ □

The finitary logical system $EC^{\hat{p}}$ is defined by the same axiom schemas as EC^{ω}. The two systems do not, however, have the same axioms since they are defined for different languages – the extensions of the schemas are different. The derivation relation for $EC^{\hat{p}}$ is defined by the axioms and the derivation rule modus ponens. Particularly, the infinitary derivation clause **R*** from EC^{ω} is not included.

Definition 17 ($EC^{\hat{p}}$) $EC^{\hat{p}}$ is the logical system for the language $\mathcal{L}^{\hat{p}}_{\bigtriangledown}$ consisting of the following axiom schemata:

All substitution instances of tautologies of propositional calculus		**Prop**
$(\bigtriangledown_i X \wedge \bigtriangledown_i Y) \to \bigtriangledown_i(X \cap Y)$		E3
$\neg(\bigtriangleup_i X \wedge \bigtriangledown_i Y)$	when $X \not\subseteq Y$	E4
$(\bigtriangledown_i(Y \cup \{\gamma\}) \wedge \neg K_i\gamma) \to \bigtriangledown_i Y$		E5
$\bigtriangledown_i X \to \bigtriangledown_i Y$	when $X \subseteq Y$	**KG**

The derivation relation $\vdash_{EC^{\hat{p}}}$ – written $\vdash_{\hat{p}}$ for simplicity – between sets of $\mathcal{L}^{\hat{p}}_{\bigtriangledown}$ formulae and single $\mathcal{L}^{\hat{p}}_{\bigtriangledown}$ formulae is the smallest relation closed under the following conditions:

$\Gamma \vdash_{\hat{p}} \phi$	when $\phi \in \Gamma$	**Prem**
$\Gamma \vdash_{\hat{p}} \phi$	when ϕ is an axiom	**Ax**
$\dfrac{\Gamma \vdash_{\hat{p}} \phi, \Gamma \vdash_{\hat{p}} \phi \to \psi}{\Gamma \vdash_{\hat{p}} \psi}$		**MP**

□

It is easy to see that E1, E2, **KS** and **DT** are derivable in EC^{ω}.

The restriction $\mathcal{L}^{\hat{p}}_{\bigtriangledown} \subset \mathcal{L}_{\bigtriangledown}$ is sufficient to prove strong completeness without **R*** in a manner very similar to the proof in Section 4.1. The first step, existence of maximal consistent extensions, can now be proved by the standard proof since the system is finitary.

Lemma 18 (Lindenbaum lemma for $\boldsymbol{EC^{\hat{p}}}$) If Γ is $EC^{\hat{p}}$-consistent, then there exists an $\mathcal{L}^{\hat{p}}_{\bigtriangledown}$-maximal and $EC^{\hat{p}}$-consistent Γ' such that $\Gamma \subseteq \Gamma'$. □

Second, we establish the result corresponding to Lemma 13 for $\mathcal{L}^{\hat{p}}_{\bigtriangledown}$ and $EC^{\hat{p}}$.

Lemma 19 Let $\Gamma' \subseteq \mathcal{L}^{\hat{p}}_{\bigtriangledown}$ be a $\mathcal{L}^{\hat{p}}_{\bigtriangledown}$-maximal and $EC^{\hat{p}}$-consistent theory. If there exists a X' such that $\Gamma' \vdash_{\hat{p}} \bigtriangledown_i X'$, then there exists a X such that $\Gamma' \vdash_{\hat{p}} \Diamond_i X$.□

Proof The proof is essentially the same as for Lemma 13, for the language $\mathcal{L}^{\hat{p}}_{\bigtriangledown}$ instead of $\mathcal{L}_{\bigtriangledown}$ (note that in that proof we did not rely on **R***, and that $\hat{p} \notin X$ since $X \subseteq X'$). ■

Third, we show satisfiability.

Lemma 20 Every maximal $EC^{\hat{p}}$-consistent $\mathcal{L}^{\hat{p}}_{\bigtriangledown}$ theory is satisfiable. □

Proof Let Γ be maximal and consistent. The proof is very similar to that of the corresponding result for EC^{ω} (Lemma 14). We construct the following SSA, which is intended to satisfy Γ:

$$M^{\Gamma} = (\{s\}, \sigma^{\Gamma})$$
$$\sigma^{\Gamma}(s)(p) = \mathbf{true} \Leftrightarrow \Gamma \vdash_{\hat{p}} p \text{ when } p \in \Theta$$
$$\sigma^{\Gamma}(s)(K_i\phi) = \mathbf{true} \Leftrightarrow \phi \in X_i^{\Gamma}$$

where:

$$X_i^{\Gamma} = \begin{cases} Z \text{ where } \Gamma \vdash_{\hat{p}} \Diamond_i Z & \text{if there is an } X' \text{ such that } \Gamma \vdash_{\hat{p}} \bigtriangledown_i X' \\ \{\gamma : \Gamma \vdash_{\hat{p}} K_i\gamma\} \cup \{\hat{p}\} & \text{if } \forall_{X'} \Gamma \not\vdash_{\hat{p}} \bigtriangledown_i X' \text{ and } \bigcup_{\Gamma \vdash_{\hat{p}} \triangle_i Y} Y \text{ is finite} \\ \{\gamma : \Gamma \vdash_{\hat{p}} K_i\gamma\} & \text{if } \forall_{X'} \Gamma \not\vdash_{\hat{p}} \bigtriangledown_i X' \text{ and } \bigcup_{\Gamma \vdash_{\hat{p}} \triangle_i Y} Y \text{ is infinite} \end{cases}$$

The existence of Z is guaranteed by Lemma 19, and, again, we show, by structural induction over ϕ, that

$$(M^{\Gamma}, s) \models \phi \Longleftrightarrow \Gamma \vdash_{\hat{p}} \phi \tag{3}$$

for all $\phi \in \mathcal{L}^{\hat{p}}_{\bigtriangledown}$. As in the proof of Lemma 14 we only show the epistemic base cases. For each base case we consider the situations when

a) there is an X' such that $\Gamma \vdash_{\hat{p}} \bigtriangledown_i X'$ or
b) $\Gamma \not\vdash_{\hat{p}} \bigtriangledown_i X'$ for every X'

corresponding to the first and to the second and third cases in the definition of X_i^{Γ}, respectively.

- $\phi = K_i\psi$: $(M^{\Gamma}, s) \models K_i\psi$ iff $\psi \in X_i^{\Gamma}$.
  $\Rightarrow$) Let $\psi \in X_i^{\Gamma}$. In case a), $X_i^{\Gamma} = Z$ where $\Gamma \vdash_{\hat{p}} \Diamond_i Z$ and by **KS**, $\Gamma \vdash_{\hat{p}} K_i\psi$. In case b), $\psi \neq \hat{p}$ (since $K_i\psi \in \mathcal{L}^{\hat{p}}_{\bigtriangledown}$) and thus $\Gamma \vdash_{\hat{p}} K_i\psi$ by construction of X_i^{Γ}.

⇐) Let $\Gamma \vdash_{\hat{p}} K_i\psi$. In case a), $\Gamma \vdash_{\hat{p}} \bigtriangledown_i Z$ and thus $\psi \in Z = X_i^{\Gamma}$ by E4 and consistency of Γ. In case b), $\psi \in X_i^{\Gamma}$ by construction.

– $\phi = \bigtriangledown_i X$: $(M^{\Gamma}, s) \models \bigtriangledown_i X$ iff $X_i^{\Gamma} \subseteq X$.

⇒) Let $X_i^{\Gamma} \subseteq X$. In case a), $\Gamma \vdash_{\hat{p}} \Diamond_i Z$ where $Z = X_i^{\Gamma} \subseteq X$, so $\Gamma \vdash_{\hat{p}} \bigtriangledown_i X$ by **KG**. In case b), if $\hat{p} \in X_i^{\Gamma}$ then $\hat{p} \in X$ which is impossible since $\bigtriangledown_i X$ is a formula. But if $\hat{p} \notin X_i^{\Gamma}$ then X_i^{Γ} is infinite (by construction) which is also impossible since X is finite – thus case b) is impossible.

⇐) Let $\Gamma \vdash_{\hat{p}} \bigtriangledown_i X$. In case a), $\Gamma \vdash_{\hat{p}} \triangle_i Z$ and by E4 and consistency of Γ $X_i^{\Gamma} = Z \subseteq X$. Case b) is impossible by definition. ∎

Theorem 21 *$EC^{\hat{p}}$ is a sound and strongly complete axiomatization of standard syntactic assignments, in the language $\mathcal{L}_{\bigtriangledown}^{\hat{p}}$.* □

Proof Soundness follows from the soundness of EC^{ω} and the fact that $\Gamma \vdash_{\hat{p}} \phi$ implies $\Gamma \vdash_{\omega} \phi$, the latter which can be seen by induction on the length of a proof in $EC^{\hat{p}}$ (every $\mathcal{L}_{\bigtriangledown}^{\hat{p}}$ formula is also a $\mathcal{L}_{\bigtriangledown}$ formula): the base case **Prem** follows by R1 (Lemma 9), the base case **Ax** follows by **Ax** and **W**, and the inductive case **MP** follows by **MP**, **W** and **Cut**. Strong completeness follows from Lemmas 20 and 18. ∎

5 Discussion

We introduced a "knows at most" operator in order to increase the expressiveness of the epistemic language with respect to standard syntactic assignments, and investigated strong axiomatization of the resulting logic. The new operator destroyed semantic compactness and thus the possibility of a strongly complete finitary axiomatization, but we presented a strongly complete infinitary axiomatization. An interesting result is that we have a strongly complete finitary axiomatization if we make the assumption that the agents can know something which is not expressible in the logical language. The results are a contribution to the logical foundation of multi-agent systems.

Related work include classical syntactic treatment of knowledge [6–8, 1], modelled in a possible worlds framework by [1] as described in Section 2. The $\bigtriangledown_i$ operator is new in the context of syntactic models. It is, however, similar to Levesque's *only knowing* operator **O** [9]. $\mathbf{O}\alpha$ means that the agent does not know more than α, but knowledge in this context means knowledge closed under logical consequence and "only knowing α" is thus different from "knowing at most" a general set of formulae. The relation between these concepts is an interesting possibility for future work.

In [4] we investigate the special case of agents who can know only *finitely* many syntactic formulae at the same time. Completeness results for such finitely restricted agents build upon the results presented in this paper. Another possibility for future work is to study other special classes of SSAs.

In this paper we have only studied the *static* aspect of syntactic knowledge. In [10], we discuss how syntactic knowledge can evolve as a result of reasoning and communication, i.e. a *dynamic* aspect of knowledge.

References

1. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning About Knowledge. The MIT Press, Cambridge, Massachusetts (1995)
2. Meyer, J.J.C., van der Hoek, W.: Epistemic Logic for AI and Computer Science. Cambridge University Press, Cambridge, England (1995)
3. Ågotnes, T., Walicki, M.: A logic for reasoning about agents with finite explicit knowledge. In Tessem, B., Ala-Siuru, P., Doherty, P., Mayoh, B., eds.: Proc. of the 8th Scandinavian Conference on Artificial Intelligence. Frontiers in Artificial Intelligence and Applications, IOS Press (2003) 163–174
4. Ågotnes, T., Walicki, M.: Complete axiomatizations of finite syntactic epistemic states. In: To appear in the Proceedings of the 3rd International Workshop on Declarative Agent Languages and Technologies (DALT 2005), Utrecht, the Netherlands (2005)
5. de Lavalette, G.R., Kooi, B., Verbrugge, R.: Strong completeness for propositional dynamic logic. In Balbiani, P., Suzuki, N.Y., Wolter, F., eds.: Preliminary Proceedings of AiML2002, Institut de Recherche en Informatique de Toulouse IRIT (2002) 377–393
6. Eberle, R.A.: A logic of believing, knowing and inferring. Synthese **26** (1974) 356–382
7. Moore, R.C., Hendrix, G.: Computational models of beliefs and the semantics of belief sentences. Technical Note 187, SRI International, Menlo Park, CA (1979)
8. Halpern, J.Y., Moses, Y.: Knowledge and common knowledge in a distributed environment. Journal of the ACM **37** (1990) 549–587
9. Levesque, H.J.: All I know: a study in autoepistemic logic. Artificial Intelligence **42** (1990) 263–309
10. Ågotnes, T., Walicki, M.: Syntactic knowledge: A logic of reasoning, communication and cooperation. In: Proceedings of the Second European Workshop on Multi-Agent Systems (EUMAS), Barcelona, Spain (2004)

Logical Spaces in Multi-Agent Only Knowing Systems

Bjørnar Solhaug[1,2] and Arild Waaler[3,4]

[1] SINTEF ICT, Norway
[2] Dep. of Information Science and Media Studies, University of Bergen, Norway
[3] Finnmark College, Norway
[4] Dep. of Informatics, University of Oslo, Norway

Abstract. We present a weak multi-agent system of Only knowing and an analysis of the logical spaces that can be defined in it. The logic complements the approach to generalizing Levesque‘s All I Know system made by Halpern and Lakemeyer. A novel feature of our approach is that the logic is defined entirely at the object level with no reference to meta-concepts in the definition of the axiom system. We show that the logic of Halpern and Lakemeyer can be encoded in our system in the form of a particular logical space.

1 Introduction

Multi-agent belief logics can be viewed as systems designed for the representation of representations (or languages) that agents use for reasoning about other agents' cognitive states. A multi-agent *only knowing system* has language constructs for representing upper and lower bounds of beliefs; it thereby has constructs for expressing the exact content of an agent's belief state. A variety of multi-modal only knowing languages have been analyzed in [5]. However, to represent defeasible patterns of reasoning in a multi-agent context, only knowing systems in which the underlying modal belief logic is K45 are particularly interesting. A natural way to design such systems is to generalize the only knowing system of Levesque [7] to the multi-modal case.

This is, however, a non-trivial task; the tricky part of this is hidden in an axiom (which we shall refer to as the $\Diamond$-axiom) to the effect that $\Diamond\varphi$ (φ is logically possible) is an axiom *for each satisfiable, objective* φ ("objective" because it does not contain any modal operators). In a series of papers [2–4, 6] Halpern and Lakemeyer have attempted to formulate an appropriate generalization of this axiom in a multi-modal language; in the solution they end up with they enrich the object language with constructs for coding the satisfiability relation into the system. They also provide their analysis with a canonical model semantics. This semantics has, however, limited power, since the only model they allow is defined on the uncountable set of all maximally consistent sets.

In [12] the second author introduced another generalization of the only knowing system of Levesque [7] to the multi-modal case which does not use meta-language operators. The $\Diamond$-axiom is instead generalized to the statement that

$\Diamond_a\varphi$ is an axiom for each *consistent a-objective* φ (a-objective because any occurrence of an a-modal operator is within the scope of a b-modal operator, $a \neq b$). He proves consistency of the system L_I by proving that a complete subset of the language has a cut-free sequent calculus. He also proves that L_I is indeed equivalent to the system that Halpern and Lakemeyer claim is the correct multi-modal generalization of Levesque's system for the common part of the languages (i.e. formulae without the meta-concept operators).

We propose yet another solution based on a constructive explication of the concept of a logical space. Intuitively, the function of a logical space is to explicate every logical possibility of the logic. This is what has been done in the single-agent only knowing system Æ presented in the paper by Lian et al. [8] and further analyzed in [11]. A logical space in Æ is a formula λ such that for each purely Boolean φ, either $\lambda \vdash \Diamond\varphi$ or $\lambda \vdash \neg\Diamond\varphi$. Compared to the only knowing system of Levesque, the system Æ has an increased expressive power due to the possibility of varying the logical space, where the $\Diamond$-axiom of Levesque corresponds to only one of many possibilities.

This paper provides a generalization of the system Æ to the multi-agent case, and hence also provides a solution to the problem with the generalized $\Diamond$-axiom. In the single-agent case, the set of possibilities is derived from a set of formulae from the language of propositional logic. The formulae of this particular set are referred to as *atoms*. Where α is an atom describing a logical possibility, the logical space λ is defined such that $\Diamond\alpha$ is entailed by λ. In the multi-agent case, we aim at defining a logical space λ_a for each agent a, such that $\Diamond_a\varphi$ is entailed by λ_a for each φ representing a logical possibility to agent a.

By providing logical spaces as a solution to the problem with the $\Diamond$-axiom, we need not encode meta-concepts into the language, nor refer to such concepts in the definition of the axiom system.

In order to bring the task of defining a multi-agent logical space to a manageable level, we will address the problem inductively at different levels of complexity, each level corresponding to a sub-language within which the set of possibilities is outlined. The base case is equivalent to the single agent case: Let $\mathcal{L}_0$ denote the language of propositional logic. The set of possibilities is derived by closing a subset of $\mathcal{L}_0$ under the $\Diamond_a$-operator for each agent a. The resulting set of formulae is then a subset of the language of the next level, denoted $\mathcal{L}_1$. Inductively, the set of possibilities for agent a at level $k+1$ is derived from a subset of *the a-objective formulae of* $\mathcal{L}_k$.

The main task of this paper is to construct the sets of formulae that, for each agent and each language level, express each and every logical possibility. We propose this as a replacement of the $\Diamond$-axiom of Levesque. In Sect, 4, we will show how the logic of $Æ_I$ applies to examples from the paper of Halpern and Lakemeyer [4]. In Sect. 6, we prove the equivalence between the systems $Æ_I$ and L_I, and hence the equivalence between the system of Halpern and Lakemeyer [4], where a particular logical space is added to the axioms of $Æ_I$.

In [13] a modal reduction property for L_I is established, which states that any "only knowing" expression is provably equivalent to a disjunction of "only knowing" expressions of a particular simple form. Each of these latter expressions provides us with an explicit syntactical representation of a particular model of the original formula. The latter expressions explicitly characterize the possible cognitive states of the agent, given the initial "only knowing" expression. In Sect. 5 we shall see that the same property holds also in $Æ_I$.

2 The Logic $Æ_I$

2.1 Syntax

The object language $\mathcal{L}$ contains a countable set of propositional letters $\mathcal{P}$, the propositional constant $\bot$, the Boolean connectives $\neg$ and $\wedge$ and the modal operators $\mathbf{B}_a$ and $\mathbf{C}_a$ for each a in a countable non-empty set of indices I. The index set I represents the set of agents, $\mathbf{B}_a$ is a belief operator, and $\mathbf{C}_a$ is a complementary *co-belief* operator for agent $a \in I$. The propositional constant $\top$ is defined as $\neg\bot$, while the Boolean connectives $\vee$, $\supset$ and $\equiv$ are the usual abbreviations. Other modal operators defined as abbreviations are the following: $\mathbf{b}_a\varphi$ (φ is compatible with belief) is $\neg\mathbf{B}_a\neg\varphi$, $\mathbf{c}_a\varphi$ (φ is compatible with co-belief) is $\neg\mathbf{C}_a\neg\varphi$, $\Box_a\varphi$ (φ is necessary) is $\mathbf{B}_a\varphi \wedge \mathbf{C}_a\varphi$ and $\Diamond_a\varphi$ (φ is possible) is $\mathbf{b}_a\varphi \vee \mathbf{c}_a\varphi$. Observe that necessity and possibility are relative to the extension of a given agent's belief and co-belief; the notion of necessity hence captures *personal* necessity.

The more accurate interpretation of the $\mathbf{B}_a$-operator is that a formula $\mathbf{B}_a\varphi$ states that agent a believes at least φ to be true, but perhaps more. The $\mathbf{B}_a$-operator thus puts a *lower* bound on the extension of belief. The complementary operator $\mathbf{C}_a$ puts an *upper* bound on the belief in the sense that a formula $\mathbf{C}_a\varphi$ states that agent a believes at most φ to be false, but perhaps less. The formula $\mathbf{B}_a\varphi \wedge \mathbf{C}_a\neg\varphi$ states that φ is *exactly* what is believed. The introduction of the $\mathbf{C}_a$-operator thus allows an "All a knows"-proposition $\mathbf{O}_a\varphi$ to be defined as $\mathbf{B}_a\varphi \wedge \mathbf{C}_a\neg\varphi$.

A formula not mentioning any modal operators is called *purely Boolean.* φ is an *a-modal atom* if it is of the form $\mathbf{B}_a\varphi$ or $\mathbf{C}_a\varphi$, $a \in I$. An a-modal *literal* is an a-modal atom or the negation of an a-modal atom. φ is a *completely a-modalized formula* if it is a Boolean combination of a-modal atoms. φ is *free of modality a* if it is a Boolean combination of propositional letters and modal atoms not of modality a. φ is a *first-order formula* if, for each $a \in I$ and each subformula $\mathbf{B}_a\psi$ and $\mathbf{C}_a\psi$ in φ, ψ is free of modality a. If Γ is a set of formulae, $\Gamma^{\setminus a} = \{\varphi \in \Gamma \mid \varphi \text{ free of modality } a\}$ and $\Gamma^a = \{\varphi \in \Gamma \mid \varphi \text{ completely } a\text{-modalized}\}$. If Γ is a set of formulae, $\mathsf{Sf}(\Gamma)$ denotes the set of subformulae of the formulae in Γ. When Γ is a singleton set containing φ, $\mathsf{Sf}(\varphi)$ denotes $\mathsf{Sf}(\{\varphi\})$.

The *modal depth* $\mathsf{d}(\varphi)$ of a formula φ expresses the nesting of alternating modalities in φ. Formally, the modal depth of a purely Boolean φ is 0. Otherwise, if φ is $\mathbf{B}_a\psi$ or $\mathbf{C}_a\psi$, let Ψ be the set of modal atoms which occur as subformulae in ψ. Then $\mathsf{d}(\varphi)$ is the maximal number in $\{\mathsf{d}(\chi) + 1 \mid \chi \in \Psi \text{ and } \chi \text{ is not } a\text{-modalized}\} \cup \{\mathsf{d}(\chi) \mid \chi \in \Psi \text{ and } \chi \text{ is } a\text{-modalized}\}$. Otherwise, the modal depth of φ is the maximal $\mathsf{d}(\psi)$ for a subformula ψ of φ. The modal depth of an a-modal formula φ is hence increased by prefixing φ with any other modal operator than an a-modal operator.

If Γ is a set of formulae, $\Gamma_k = \{\varphi \in \Gamma \mid \mathsf{d}(\varphi) \leq k\}$. We will in this paper be interested in sub-languages relative to a given modal depth and a given agent. Since $\mathcal{L}$ denotes the language of $Æ_I$ (which is just a set of formulae) these sublanguages are denoted $\mathcal{L}_k$, $\mathcal{L}_k^{\setminus a}$ and $\mathcal{L}_k^a$ following the set indexing notation introduced above.

A *tautology* is a substitution instance of a formula valid in propositional logic, e.g. $\Box_a\varphi \supset \Box_a\varphi$. The deducibility relation '$\vdash$' of the logic $Æ_I$ is defined as the least relation that contains all tautologies, is closed under all instances of the rules

$$\frac{\vdash \varphi}{\vdash \Box_a\varphi}\ \text{(RN)} \qquad \frac{\vdash \varphi \quad \vdash \varphi \supset \psi}{\vdash \psi}\ \text{(MP)}$$

and contains all instances of the following schemata for each $a \in I$:

$K_{\mathbf{B}}$:	$\mathbf{B}_a(\varphi \supset \psi) \supset (\mathbf{B}_a\varphi \supset \mathbf{B}_a\psi)$	$\overline{B}_\Box$:	$\neg\mathbf{B}_a\varphi \supset \Box_a\neg\mathbf{B}_a\varphi$
$K_{\mathbf{C}}$:	$\mathbf{C}_a(\varphi \supset \psi) \supset (\mathbf{C}_a\varphi \supset \mathbf{C}_a\psi)$	$\overline{C}_\Box$:	$\neg\mathbf{C}_a\varphi \supset \Box_a\neg\mathbf{C}_a\varphi$
$B_\Box$:	$\mathbf{B}_a\varphi \supset \Box_a\mathbf{B}_a\varphi$	T:	$\Box_a\varphi \supset \varphi$
$C_\Box$:	$\mathbf{C}_a\varphi \supset \Box_a\mathbf{C}_a\varphi$		

We write $\vdash \varphi$ if φ is theorem of $Æ_I$, and $\varphi_1, \ldots, \varphi_n \vdash \psi$ for $\vdash (\varphi_1 \wedge \cdots \wedge \varphi_n) \supset \psi$. $\Gamma \vdash \varphi$ means that there is a finite number of formulae $\gamma_1, \ldots, \gamma_n$ in Γ such that $\gamma_1, \ldots, \gamma_n \vdash \varphi$. If $\Gamma \vdash \bot$, Γ is *inconsistent* otherwise Γ is consistent. We will without reference use the well-known principles of modal logic, especially substitution of provable equivalents, the derived rule

$$\frac{\varphi_1, \ldots, \varphi_n \vdash \psi}{\mathbf{B}_a\varphi_1, \ldots, \mathbf{B}_a\varphi_n \vdash \mathbf{B}_a\psi}$$

and the corresponding rule for $\mathbf{C}_a$.

Lemma 1. *$\Box_a$ is an S5 modality.*

Lemma 2. *Any formula is provably equivalent to a first-order formula with the same modal depth.*

The former of these two results is Lemma 1 of [13]; the latter is Lemma 2 of [12]. For proofs and further details about the results in the rest of this section, the reader may consult [13].

2.2 Semantics

A *frame* is a structure $(W, \{R_a, S_a \mid a \in I\})$, where W is a non-empty set of points and R_a and S_a are binary relations satisfying the following two conditions:

$(f1)$ Let X be either R_a or S_a and Y be either R_a or S_a or their complements $\overline{R_a}$ or $\overline{S_a}$. Then the composition $X \circ Y \subseteq Y$.
$(f2)$ $E_a = R_a \cup S_a$ is reflexive.

Note that in standard terminology two of the eight subconditions of $(f1)$ state that R_a and S_a are transitive, e.g. $R_a \circ R_a \subseteq R_a$, while two of them state that they are Euclidean, e.g. $R_a \circ \overline{R_a} \subseteq \overline{R_a}$.

Lemma 3. *E_a is an equivalence relation.*

An *a-cluster* is an equivalence class of W modulo E_a. Let C be an a-cluster. We define the *belief part* C^+ and the *co-belief part* C^- of C by: $C^+ = \{x \in C \mid xR_ax\}$ and $C^- = \{x \in C \mid xS_ax\}$. C is *bisected* if $C^+ \cap C^- = \emptyset$.

Lemma 4. $C = C^+ \cup C^-$.

A *model* $\mathcal{M} = (W, \{R_a, S_a \mid a \in I\}, V)$ is a frame with a valuation function V, which maps each propositional letter onto a subset of W. The satisfiability relation $\vDash_x$, $x \in W$, is defined by

$$\begin{aligned}
&\mathcal{M} \vDash_x p && \leftrightarrow x \in V(p),\ p \text{ a propositional letter,}\\
&\mathcal{M} \vDash_x \neg\varphi && \leftrightarrow \mathcal{M} \nvDash_x \varphi,\\
&\mathcal{M} \vDash_x \mathbf{B}_a\varphi && \leftrightarrow \forall y\, (xR_ay \to \mathcal{M} \vDash_y \varphi),\\
&\mathcal{M} \vDash_x \mathbf{C}_a\varphi && \leftrightarrow \forall y\, (xS_ay \to \mathcal{M} \vDash_y \varphi),
\end{aligned}$$

and in the usual way for the other Boolean connectives. We write $\mathcal{M} \models_X \varphi$ iff $(\forall x \in X)(\mathcal{M} \models_x \varphi)$. A formula is valid in a frame if it is true at all points in all models on the frame. If φ is valid in all frames, we write $\vDash \varphi$, and say that φ is *valid*. $\Gamma \models \varphi$ means that for all models, φ is true at all points which satisfy all formulae in Γ. Note that if C is an a-cluster, all points in C agree on every completely a-modalized formula in every model on the frame.

Theorem 1. *Æ$_I$ is sound, complete and decidable.*

Proof. This can be proved by the use of standard techniques from modal logic, see [13]. □

3 Finitely Bound Sublanguages

The syntax and semantics of the previous section generalize the syntax and semantics of the system Æ to the multi-agent case. What remains to be done is to generalize the notion of a logical space to the multi-agent case. The properties of the single-agent logical space will serve as guiding principles for our multi-agent generalization.

In the single-agent case, under the assumption that there are finitely many propositional letters in the language, say $p_1, \dots, p_m$, an *atom* is defined as a conjunction $\pm p_1 \wedge \cdots \wedge \pm p_m$, where $\pm p$ means either p or $\neg p$. An atom can be interpreted as characterizing the material content of a state of affairs. There are 2^m atoms. Where $\alpha_1, \dots, \alpha_n$ characterize the conceivable states of affairs, the logical space is defined as the formula

$$\Diamond\alpha_1 \wedge \cdots \wedge \Diamond\alpha_n \wedge \Box(\alpha_1 \vee \cdots \vee \alpha_n).$$

In the *maximal* logical space, all atoms are possible, i.e. $n = 2^m$.

The notion of a logical space is a notion of personal necessity. To see this, observe that we may define a logical space such that concepts that intuitively are logically independent are related in the logic at the level of necessity. We may e.g. define a logical space λ such that $\lambda \vdash \Box(\text{penguin}(\text{Tweety}) \supset \text{bird}(\text{Tweety}))$.

In order to define a multi-agent logical space, we need to generalize the notion of an atom. To do this, we first define a *finite multi-modal language*. For such a language to be finite, the set of propositional letters and the set of different modalities, i.e. the index set I, obviously need to be finite. Furthermore, as we may construct new formulae by prefixing a formula free of modality a with any a-modal operator, the finite language must be bound by a limit on modal depth. Under the assumption that the set of propositional letters and the set of agents are finite, we will for each modal depth k operate with the sublanguage $\mathcal{L}_k$.

The notion of an atom is a twin to the notion of a *complete theory*. Given any language $\mathcal{L}^*$, a formula $\varphi \in \mathcal{L}^*$ is a complete theory for $\mathcal{L}^*$ iff for all formulae $\psi \in \mathcal{L}^*$, either $\varphi \vdash \psi$ or $\varphi \vdash \neg\psi$. The modal language of Æ, denoted $\mathcal{L}_1$ for a singleton set I, can be seen as representations of formulae from $\mathcal{L}_0$, i.e. the language of propositional logic. As an atom can be interpreted as a propositional valuation, it is easy to see that each atom is a complete theory for $\mathcal{L}_0$. Generalizing this to the multi-agent case is to say that a multi-modal language is a language representing representations in a modal language, the first capturing the cognitive state of a given agent a, the latter representing the material content of states of affairs as well as the cognitive state of every agent different from a. The latter language is then denoted by $\mathcal{L}_k^{\setminus a}$ for some given integer k, the former the closure of $\mathcal{L}_k^{\setminus a}$ under the a-modal operators. A complete theory for $\mathcal{L}_k^{\setminus a}$ is thus a complete characterization of the material content of a state of affairs in addition to a complete characterization of the cognitive state of every agent different from a given agent a.

The notion of a complete theory may then serve as a test for deciding whether a suggested multi-modal logical space is a correct generalization of the single-agent logical space. In other terms, if $\Diamond_a\varphi_1 \wedge \cdots \wedge \Diamond_a\varphi_n \wedge \Box_a(\varphi_1 \vee \cdots \vee \varphi_n)$ is a multi-modal logical space for a given agent a, where $\{\varphi_1, \dots, \varphi_n\} \subseteq \mathcal{L}_k^{\setminus a}$ for a given integer k, then each $\varphi \in \{\varphi_1, \dots, \varphi_n\}$ should be a complete theory for $\mathcal{L}_k^{\setminus a}$.

We will use the following notation for the distribution of a modality over a set of formulae: $\mathbf{B}_a\Gamma = \{\mathbf{B}_a\gamma \mid \gamma \in \Gamma\}$, and the same for any other modality.

Definition 1. *Let $\Phi \subseteq \mathcal{L}^{\setminus a}$. The functions* Bel_a, Cobel_a *and* Lspace_a*, all of them from a set of formulae free of modality a to a completely a-modalized formula, are defined as follows:*

$$\mathsf{Bel}_a(\Phi) = \bigwedge \boldsymbol{b}_a\Phi \wedge \boldsymbol{B}_a(\bigvee \Phi),$$
$$\mathsf{Cobel}_a(\Phi) = \bigwedge \boldsymbol{c}_a\Phi \wedge \boldsymbol{C}_a(\bigvee \Phi),$$
$$\mathsf{Lspace}_a(\Phi) = \bigwedge \Diamond_a\Phi \wedge \Box_a(\bigvee \Phi).$$

$\mathsf{Lspace}_a(\Phi)$ *is* the logical space *for agent a* spanned by Φ. *If $\Phi^+ \cup \Phi^- = \Phi$, then* $\mathsf{Bel}_a(\Phi^+) \wedge \mathsf{Cobel}_a(\Phi^-)$ *is a* doxastic a-alternative *spanned by Φ. Notice that a doxastic a-alternative spanned by Φ entails* $\mathsf{Lspace}_a(\Phi)$*. The set of all doxastic a-alternatives spanned by every nonempty subset of Φ is denoted* $\mathsf{Dox}_a(\Phi)$*.*

Two properties will play a central role in our analysis. A set of formulae Φ satisfies the *strong independence property* if every two elements of Φ are consistent iff they are equivalent. Φ is $\mathcal{L}^*$*-saturated* if $\Phi \subseteq \mathcal{L}^*$ and every formula $\varphi \in \mathcal{L}^*$ is equivalent to a disjunction of formulae in Φ ($\mathcal{L}^*$ any language addressed in this paper).

Lemma 5. *If $\Phi \subseteq \mathcal{L}^{\setminus a}$ satisfies strong independence, then so does* $\mathsf{Dox}_a(\Phi)$*.*

Proof. Let δ_1 and δ_2 be two distinct elements of $\mathsf{Dox}_a(\Phi)$. Then δ_1 and δ_2 must disagree on the belief set or the co-belief set. We treat the former. Let $\delta_1 \vdash \mathsf{Bel}_a(\Gamma_1)$ and $\delta_2 \vdash \mathsf{Bel}_a(\Gamma_2)$. There is then a formula φ such that either $\varphi \in \Gamma_1$ and $\varphi \notin \Gamma_2$ or vice verca. In the first case, $\varphi \wedge \gamma \vdash \bot$ for each $\gamma \in \Gamma_2$ by strong independence. Hence $\varphi \wedge \bigvee \Gamma_2 \vdash \bot$. By modal logic, $\mathbf{B}_a(\bigvee \Gamma_2) \vdash \mathbf{B}_a\neg\varphi$. Since $\delta_2 \vdash \mathbf{B}_a(\bigvee \Gamma_2)$ and $\delta_1 \vdash \mathbf{b}_a\varphi$, we get $\delta_1 \wedge \delta_2 \vdash \bot$. The latter case is symmetrical. □

Lemma 6. *Let $\mathcal{L}^*$ be any Boolean closed set of formulae and Φ be $\mathcal{L}^*$-saturated. Then $\vdash \bigvee \Phi$.*

Proof. Assume that $\neg \bigvee \Phi$ is consistent. Since Φ is $\mathcal{L}^*$-saturated and $\mathcal{L}^*$ is Boolean closed, there must then be a non-empty set $\Gamma \subseteq \Phi$ such that $\vdash \bigvee \Gamma \equiv \neg \bigvee \Phi$. But this is clearly impossible. □

Lemma 7. *Let Φ be $\mathcal{L}_k^{\setminus a}$-saturated. Then* $\mathsf{Dox}_a(\Phi)$ *is $\mathcal{L}_{k+1}^a$-saturated.*

Proof. Let $\varphi \in \mathcal{L}_{k+1}^a$. We may without loss of generality assume that φ is first-order. Since Φ is $\mathcal{L}_k^{\setminus a}$-saturated, the formulae inside the scope of the a-modalities are equivalent to disjunctions of formulae from Φ. By standard propositional reasoning and normal modal logic and, φ is equivalent to a formula on DNF, where each disjunct is of the form $\psi = \bigwedge \mathbf{b}_a\Gamma_1 \wedge \mathbf{B}_a(\bigvee \Gamma_2) \wedge \bigwedge \mathbf{c}_a\Gamma_3 \wedge \mathbf{C}_a(\bigvee \Gamma_4)$, $\Gamma_1, \ldots, \Gamma_4$ subsets of Φ. Let

$$\Delta = \{\delta \in \mathsf{Dox}_a(\Phi) \mid \delta = \mathsf{Bel}_a(\Phi^+) \wedge \mathsf{Cobel}_a(\Phi^-), \Gamma_1 \subseteq \Phi^+ \subseteq \Gamma_2, \Gamma_3 \subseteq \Phi^- \subseteq \Gamma_4\}.$$

Then $\vdash \psi \equiv \bigvee \Delta$. To see that $\bigvee \Delta \vdash \psi$, observe that if $\Gamma_1 \subseteq \Phi^+$, then $\mathsf{Bel}_a(\Phi^+) \vdash \mathbf{b}_a\gamma$ for each $\gamma \in \Gamma_1$, and if $\Phi^+ \subseteq \Gamma_2$, then $\mathbf{B}_a(\bigvee \Phi^+) \vdash \mathbf{B}_a(\bigvee \Gamma_2)$. Conversely, assume that $\psi \nvdash \bigvee \Delta$, i.e. that ψ is consistent with $\neg \bigvee \Delta$. This entails that ψ is consistent with a formula θ constructed as a conjunction out of the negation of one conjunct from each δ in Δ. But by construction of ψ there is no such θ which is consistent with ψ. □

Lemma 8. *Let Φ be $\mathcal{L}_k^{\setminus a}$-saturated and satisfy strong independence, and let δ be a doxastic a-alternative spanned by $\Gamma \subseteq \Phi$. Then δ is a complete theory over $\mathcal{L}_{k+1}^a$.*

Proof. We need to prove that either $\delta \vdash \psi$ or $\delta \vdash \neg\psi$ for every $\psi \in \mathcal{L}_{k+1}^a$. By Lemma 2, we may without loss of generality assume that ψ is first-order. The result for Boolean combinations of formulae follows easily once the result is established for modal atoms. It suffices to deal with the case where ψ is of the form $\mathbf{B}_a\varphi$, as the other cases are symmetrical.

Let $\delta \vdash \mathsf{Bel}_a(\Gamma_1)$ and $\vdash \varphi \equiv \bigvee \Gamma_2$, Γ_1 and Γ_2 subsets of Φ. There are two cases. Either $\Gamma_1 \subseteq \Gamma_2$, or there is a formula γ such that $\gamma \in \Gamma_1$ and $\gamma \notin \Gamma_2$. In the first case, $\bigvee \Gamma_1 \vdash \bigvee \Gamma_2$. By modal logic, $\mathbf{B}_a(\bigvee \Gamma_1) \vdash \mathbf{B}_a(\bigvee \Gamma_2)$, and so $\delta \vdash \mathbf{B}_a\varphi$. In the second case, $\gamma \wedge \bigvee \Gamma_2 \vdash \bot$ by strong independence. By modal logic, $\mathbf{b}_a\gamma \vdash \mathbf{b}_a\neg(\bigvee \Gamma_2)$. Since $\delta \vdash \mathbf{b}_a\gamma$, we get that $\delta \vdash \mathbf{b}_a\neg(\bigvee \Gamma_2)$, i.e. $\delta \vdash \neg\mathbf{B}_a\varphi$. □

We are now ready to generalize the single-agent notion of an atom to the multi-agent case. In the single agent case, an atom α can be interpreted as a complete characterization of the material content of a state of affairs. In the multi-agent case, we want for each agent $a_i \in I$, $I = \{a_1, \ldots, a_m\}$, and each modal depth k to define a doxastic alternative δ_i, such that δ_i completely characterizes the cognitive state of agent a_i. A conjunction $\alpha \wedge \delta_1 \wedge \cdots \wedge \delta_m$ is then a complete characterization of the material content of a state of affairs, as well as a complete characterization of the cognitive state of every agent. As we shall see, the conjunction $\alpha \wedge \delta_1 \wedge \cdots \wedge \delta_m$ is a complete theory for $\mathcal{L}_k$. This conjunction will be referred to as *an I-atom with depth k*.

Given a set of I-atoms with depth k, the doxastic alternatives for agent a with depth $k+1$ will be defined over this set. Intuitively, where Φ is the set of I-atoms with depth k, the set of formulae $\mathsf{Dox}_a(\Phi)$ is the set of doxastic a-alternatives with depth $k+1$. This is, however, not the correct generalization of the single agent case, since in the single-agent case, a doxastic alternative is defined over a set of *purely Boolean formulae.* Generalizing this is to define a doxastic alternative for agent a_i over a set of formulae *free of modality a.* To this end, we will define a set of formulae from $\mathcal{L}_k^{\setminus a}$ each formula of which forms a complete theory for $\mathcal{L}_k^{\setminus a}$.

Convention. Let $\varphi = \alpha \wedge \delta_1 \wedge \cdots \wedge \delta_m$ be a formula such that $\alpha \in \mathcal{L}_0$ and δ_i is a doxastic a-alternative. Then $\varphi[a_i/\top] = \alpha \wedge \delta_1 \wedge \cdots \wedge \delta_{i-1} \wedge \top \wedge \delta_{i+1} \wedge \cdots \wedge \delta_m$. If Φ is a set of I-atoms, $\Phi[a_i/\top] = \{\varphi[a_i/\top] \mid \varphi \in \Phi\}$.

Definition 2 (I-atoms). *The set of I-atoms Φ_k with depth k is defined as follows: Φ_0 is the set of atoms, while Φ_{k+1} is all formulae $\alpha \wedge \delta_1 \wedge \cdots \wedge \delta_m$ such that*

- *α is an atom,*
- *δ_i is a doxastic a_i-alternative spanned by the set $\Gamma_i \subseteq \Phi_k[a_i/\top]$,*
- *$\exists \varphi \in \Phi_k$ such that $\varphi \vdash \alpha$ and for each a_i, $\varphi[a_i/\top] \in \Gamma_i$.*

From now on Φ_k refers to the set of I-atoms with depth k. The third condition in the definition above is a consistency condition as witnessed by the following result.

Lemma 9. *Assume that each Φ_k satisfies strong independence and that $\Phi_k[a/\top]$ is $\mathcal{L}_k^{\setminus a}$-saturated for each agent a. Let α be an atom and δ_i be a doxastic a_i-alternative spanned by $\Gamma_i \subseteq \Phi_k[a_i/\top]$. Then $\alpha \wedge \delta_1 \wedge \cdots \wedge \delta_m$ is consistent if and only if $\exists \varphi \in \Phi$ such that $\varphi \vdash \alpha$ and for each a_i, $\varphi[a_i/\top] \in \Gamma_i$.*

Proof. Note that if δ_i is spanned by the set $\Gamma_i \subseteq \Phi_k[a_i/\top]$, then $\delta_i \vdash \Box_{a_i}(\bigvee \Gamma_i)$. By axiom T, $\delta_i \vdash \bigvee \Gamma_i$. Since the conjuncts of ψ are of different modalities (the atom purely Boolean, however), inconsistency of ψ can stem from axiom T only. Hence, it suffices to prove that the consistency condition ensures consistency of $\psi = \alpha \wedge \bigvee \Gamma_1 \wedge \cdots \wedge \bigvee \Gamma_m$.

Also note that Φ_0 is the set of atoms (which is trivially $\mathcal{L}_0$-saturated) and that the condition for $k = 1$ then simply states that there is an atom α such that $\alpha \in \Gamma_i$ for each Γ_i. If there is a Γ_i such that $\alpha \notin \Gamma_i$, Lemma 6 gives that $\bigvee \Gamma_i \vdash \neg\alpha$. Hence ψ is inconsistent. Conversely, if ψ is inconsistent, there must be a Γ_i such that $\alpha \notin \Gamma_i$, and hence the condition is not satisfied.

If $k > 1$, suppose that the condition is not satisfied. Then, for each $\varphi \in \Phi_k$ which entails α there is a Γ_i such that $\varphi[a_i/\top] \notin \Gamma_i$. It follows from this that given any two distinct sets Γ_i and Γ_j, each two elements $\varphi_1[a_i/\top] \in \Gamma_i$ and $\varphi_2[a_j/\top] \in \Gamma_j$ must disagree on a doxastic a_l-alternative, $a_i \neq a_j \neq a_l$. In other terms, there are two distinct doxastic a_l-alternatives δ_1 and δ_2 such that $\varphi_1[a_i/\top] \vdash \delta_1$ and $\varphi_2[a_j/\top] \vdash \delta_2$. By the strong independence assumption, $\delta_1 \wedge \delta_2 \vdash \bot$, and so $\varphi_1[a_i/\top] \wedge \varphi_2[a_j/\top] \vdash \bot$. Since this holds for any two distinct sets Γ_i and Γ_j, ψ must be inconsistent.

Suppose conversely that ψ is inconsistent. There are two cases. In the first case, there is a set Γ_i such that $\bigvee \Gamma_i \vdash \neg\alpha$, i.e. for each $\varphi \in \Phi_k$ such that $\varphi \vdash \alpha$, there is a set Γ_i such that $\varphi[a_i/\top] \notin \Gamma_i$. Then the condition is not satisfied. In the second case, there are two distinct sets Γ_i and Γ_j such that $\bigvee \Gamma_i \wedge \bigvee \Gamma_j \vdash \bot$. Then, for each two elements $\varphi_1[a_i/\top] \in \Gamma_i$ and $\varphi_2[a_j/\top] \in \Gamma_j$, $\varphi_1[a_i/\top] \wedge \varphi_2[a_j/\top] \vdash \bot$. We may assume that α is entailed by both φ_1 and φ_2 since this was treated in the first case. Since $\varphi_1[a_i/\top] \wedge \varphi_2[a_j/\top] \vdash \bot$, the two formulae must disagree on a doxastic a_l-alternative, and hence φ_1 and φ_2 are two distinct elements of Φ_k. The condition is then not satisfied. $\Box$

Lemma 10. *The set $\Phi_k[a/\top]$ satisfies strong independence.*

Proof. The base case is when $k = 0$. $\Phi_0[a/\top]$ is the set of atoms, and it is immediate that the set of atoms satisfies strong independence. Suppose inductively that φ and ψ are two distinct elements of $\Phi_{k+1}[a/\top]$. Then φ and ψ either disagree on an atom or on a doxastic b-alternative, $b \neq a$. In the first case, it is immediate that $\varphi \wedge \psi \vdash \bot$. In the second case, let $\varphi \vdash \delta_b^1$ and $\psi \vdash \delta_b^2$, where δ_b^1 and δ_b^2 are doxastic b-alternatives spanned by Γ_1 and Γ_2, respectively, Γ_1 and Γ_2 subsets of $\Phi_k[b/\top]$. By the induction hypothesis, $\Phi_k[b/\top]$ satisfies strong independence. By Lemma 5, $\delta_b^1 \wedge \delta_b^2 \vdash \bot$. Hence $\varphi \wedge \psi \vdash \bot$. □

Corollary 1. *The set of doxastic a-alternatives spanned by subsets of $\Phi_k[a/\top]$ satisfies strong independence.*

Proof. Immediate from Lemma 10 and Lemma 5. □

Lemma 11. *$\Phi_k[a/\top]$ is $\mathcal{L}_k^{\setminus a}$-saturated and $\mathsf{Dox}_a(\Phi_k[a/\top])$ is $\mathcal{L}_{k+1}^a$-saturated.*

Proof. Both properties are proved by simultaneous induction on k. In the base case $\Phi_0 = \Phi_0^{\setminus a}$. It is easy to see that the first condition holds. Since Φ_0 is $\mathcal{L}_0$-saturated, the second holds by Lemma 7.

$\Phi_{k+1}[a/\top]$ is $\mathcal{L}_{k+1}^{\setminus a}$-saturated (induction step). We have to prove that for each $\varphi \in \mathcal{L}_{k+1}^{\setminus a}$ there is a subset of $\Phi_{k+1}[a/\top]$ the disjunction of which is equivalent to φ. It is easy to see (using the DNF equivalent of each formula) that it is sufficient to prove this for φ of the form $\varphi^{\mathcal{P}} \wedge \varphi^{a_1} \wedge \cdots \wedge \varphi^{a_m}$ where $\varphi^{\mathcal{P}}$ is purely Boolean, φ^a is $\top$ and every other φ^{a_i} is in $\mathcal{L}_{k+1}^{a_i}$. Let the atom set $\widehat{\varphi}^{\mathcal{P}}$ be the set of atoms which imply $\varphi^{\mathcal{P}}$, $\widehat{\varphi}^a$ be $\{\top\}$ and $\widehat{\varphi}^{a_i}$ be the set of all $\delta \in \mathsf{Dox}_{a_i}(\Phi_k[a_i/\top])$ such that $\delta \vdash \varphi$. Let $\widehat{\varphi}$ be the set of every formula $\alpha \wedge \delta_1 \wedge \cdots \wedge \delta_m$ in $\Phi_{k+1}[a/\top]$ such that $\alpha \in \widehat{\varphi}^{\mathcal{P}}$ and $\delta_i \in \widehat{\varphi}^{a_i}$.

It follows by construction that $\bigvee \widehat{\varphi} \vdash \varphi$. Conversely, assume that φ is consistent with $\neg \bigvee \widehat{\varphi}$. By induction hypothesis and Lemma 6, $\vdash \mathsf{Dox}_{a_i}(\Phi_k[a_i/\top])$. This entails that there must be a consistent ψ of the form $\alpha \wedge \delta_1 \wedge \cdots \wedge \delta_m$ which implies φ and which is not in $\Phi_{k+1}[a/\top]$. But this is only possible if δ violates the third subcondition in the definition of Φ_{k+1} (Definition 2). By Lemma 9, ψ is inconsistent. Contradiction. Hence $\vdash \varphi \equiv \bigvee \widehat{\varphi}$.

$\mathsf{Dox}_a(\Phi_k[a/\top])$ is $\mathcal{L}_{k+1}^a$-saturated (induction step). By the induction hypothesis, $\Phi_k[a/\top]$ is $\mathcal{L}_k^{\setminus a}$-saturated. Then, by Lemma 7, $\mathsf{Dox}_a(\Phi_k[a/\top])$ is $\mathcal{L}_{k+1}^a$-saturated. □

Theorem 2. *Each formula $\varphi \in \Phi_k^{\setminus a}$ is a complete theory over $\mathcal{L}_k^{\setminus a}$. Each doxastic alternative $\delta \in \mathsf{Dox}_a(\Phi_k[a/\top])$ is a complete theory over $\mathcal{L}_{k+1}^a$.*

Proof. By Lemma 10, $\Phi_k[a/\top]$ satisfies strong independence and by Lemma 11 $\Phi_k[a/\top]$ is $\mathcal{L}_k^{\setminus a}$-saturated. By Lemma 8, each $\delta \in \mathsf{Dox}_a(\Phi_k[a/\top])$ is a complete theory over $\mathcal{L}_{k+1}^a$. Since each $\delta \in \mathsf{Dox}_a(\Phi_{k-1}[a/\top])$ is a complete theory over $\mathcal{L}_k$, it follows that each $\varphi \in \Phi_k[a/\top]$ is a complete theory over $\mathcal{L}_k^{\setminus a}$. □

Having defined the set of I-atoms, we may now define the logical space for the multi-agent case. A *logical space of agent a up to depth k* is defined over a subset Γ of $\Phi_k[a/\top]$ by the formula $\mathsf{Lspace}_a(\Gamma)$. Observe that for $k = 0$, the logical space is defined by the formula $\mathsf{Lspace}_a(\Delta)$, $\Delta \subseteq \Phi_0$, which is a logical space as defined for the single-agent system Æ.

Corollary 2. *Let λ be a logical space for agent a up to k and $\varphi \in \mathcal{L}_k^{\setminus a}$. Then either $\lambda \vdash \Diamond_a\varphi$ or $\lambda \vdash \neg\Diamond_a\varphi$.*

4 Examples

In Section 6.1 of [4], Halpern and Lakemeyer give examples of how their logic can be used to represent default reasoning in a multi-agent situation. We will show how the inferences are carried out in the logic Æ$_I$.

Example 1. The first example of Halpern and Lakemeyer is this. Let p be agent a's secret and suppose he makes the assumption that unless he believes that b knows his secret, he assumes that she does not know it. We will now prove that if this is all he believes and if it is conceivable that b does not know his secret, then he believes that she does not know his secret. Formally, we show

$$\lambda_a \wedge \mathbf{O}_a(\neg\mathbf{B}_a\mathbf{B}_bp \supset \neg\mathbf{B}_bp) \vdash \mathbf{B}_a\neg\mathbf{B}_bp,$$

where λ_a is the logical space of agent a. Let φ denote $\neg\mathbf{B}_a\mathbf{B}_bp \supset \neg\mathbf{B}_bp$. Note that the assumption that it is conceivable to a that b does not know his secret implies that $\lambda_a \vdash \Diamond_a\neg\mathbf{B}_bp$. Let us turn to the formal derivation.

1. $\lambda_a \wedge \mathbf{O}_a\varphi \vdash \mathbf{B}_a\varphi$	PL
2. $\lambda_a \wedge \mathbf{O}_a\varphi \vdash \mathbf{C}_a\neg\varphi$	PL
3. $\lambda_a \wedge \mathbf{O}_a\varphi \vdash (\mathbf{B}_a\varphi \wedge \neg\mathbf{B}_a\mathbf{B}_bp) \supset \mathbf{B}_a\neg\mathbf{B}_bp$	normal logic, Æ$_I$
4. $\lambda_a \wedge \mathbf{O}_a\varphi \vdash \mathbf{C}_a\neg\varphi \supset (\mathbf{C}_a\neg\mathbf{B}_a\mathbf{B}_bp \wedge \mathbf{C}_a\mathbf{B}_bp)$	normal logic
5. $\lambda_a \wedge \mathbf{O}_a\varphi \vdash \Diamond_a\neg\mathbf{B}_bp$	assumption
6. $\lambda_a \wedge \mathbf{O}_a\varphi \vdash \mathbf{C}_a\mathbf{B}_bp \supset \neg\mathbf{B}_a\mathbf{B}_bp$	5, PL
7. $\lambda_a \wedge \mathbf{O}_a\varphi \vdash \neg\mathbf{B}_a\mathbf{B}_bp$	2, 4, 6, PL
8. $\lambda_a \wedge \mathbf{O}_a\varphi \vdash \mathbf{B}_a\neg\mathbf{B}_bp$	1, 3, 7, PL

In the third line, we made use of the modal reductive strength of the logic. The critical point in the derivation is of course the fifth line. This theorem rests on the assumption that $\neg\mathbf{B}_bp$ is conceivable to agent a. The derivation in the system of Halpern and Lakemeyer is somewhat longer, since they need to apply some extra machinery to reason about validity and satisfiability.

The nonmonotonicity becomes apparent when we add $\mathbf{B}_bp$ to the belief set of agent a, or we define the logical space such that $\lambda_a \vdash \neg\Diamond_a\neg\mathbf{B}_bp$. Then $\mathbf{B}_a\mathbf{B}_bp$ is deducible.

Example 2. In their next example, Halpern and Lakemeyer show how one agent reasons about another agent's ability to reason nonmonotonically. The letter p

stands for "Tweety flies". It is then shown that if a believes that all b believes is that by default Tweety flies, then a believes that b believes that Tweety flies.

Again, it is the logical space that makes the deduction go through in our system. But here, since a reasons about b's ability to reason nonmonotonically, if a is to derive the conclusion that b believes p, a must believe that the conceivability space of b is such that p is conceivable. I.e., the logical space λ_b of b must be such that $\lambda_b \vdash \Diamond_b p$. Note that since the nonmonotonicity in this example is about b, we need not consult the logical space of a.

What we want to prove is thus that

$$\mathbf{B}_a(\lambda_b \wedge \mathbf{O}_b(\mathbf{b}_b p \supset p)) \vdash \mathbf{B}_a \mathbf{B}_b p.$$

In the same pattern as in the previous example, we may show that $\lambda_b \wedge \mathbf{O}_b(\mathbf{b}_b p \supset p) \vdash \mathbf{B}_b p$, given the assumption that $\lambda_b \vdash \Diamond_b p$. The difference is that we now reason about agent b and that $\neg \mathbf{B}_b p$ is replaced with p. The desired result then follows by normal logic.

The assumption we made that a believes λ_b is a stronger assumption than what we actually need. It may very well be such that a believes that b has one of several different conceivability spaces. The assumption we need is that every logical space of agent b compatible with a's beliefs must be such that $\Diamond_b p$ is implied by it.

5 The Modal Reduction Theorem

We will in this section assume a sub-language $\mathcal{L}_k$ bound by a finite set of propositional letters $\mathcal{P}$, a finite set of indices I and a given modal depth k. Let the logical space λ of agent a be given, and let β be any formula. The modal reduction theorem states that there are formulae $\beta_1, \ldots, \beta_n$ free of modality a, such that

$$\lambda \vdash \mathbf{O}_a \beta \equiv \mathbf{O}_a \beta_1 \vee \cdots \vee \mathbf{O}_a \beta_n.$$

Moreover, each formula $\mathbf{O}_a \beta_i$, $i \leq m$, is defined directly from one of the a-clusters satisfying $\lambda \wedge \mathbf{O}_a \beta$, and each such a-cluster is represented by a formula $\mathbf{O}_a \beta_i$.

Let $\lambda \wedge \mathbf{O}_a \varphi$ have depth k. We will say that $\lambda \wedge \mathbf{O}_a \varphi$ is an *explicit belief representation* if for any formula $\psi \in \mathcal{L}_k^a$, either $\lambda \wedge \mathbf{O}_a \varphi \vdash \psi$ or $\lambda \wedge \mathbf{O}_a \varphi \vdash \neg\psi$. In other terms, an explicit belief representation is a formula that determines the agent's attitude towards any formula in the language.

Theorem 3. *Let φ be any formula free of modality a. Then $\lambda \wedge \mathbf{O}_a \varphi$ is an explicit belief representation.*

Related to the notion of an explicit belief representation is the notion of an *implicit* belief representation, i.e. formulae of the form $\lambda \wedge \mathbf{O}_a \varphi$ that allow ambiguity with respect to a-modalized formulae. An implicit belief representation is a formula $\lambda \wedge \mathbf{O}_a \varphi$ where φ is not free of modality a. By applying the modal reduction theorem, such formulae are reduced to disjunctions of formulae, each of them an explicit belief representation.

6 Related Work

We will in this section prove the equivalence of the system $\AE_I$ with two earlier attempts of generalizing the system of Levesque [7]. The first of these other systems is the system $\mathcal{HL}$ of Halpern and Lakemeyer [4], where a generalization of Levesque's system is provided by coding the satisfiability relation into the system. Notice that the language of $\mathcal{HL}$ is an extension of $\mathcal{L}$. We will prove the equivalence with $\AE_I$ with respect to the common part of the languages. The second system is the system L_I of Waaler [12], where the $\Diamond$-axiom of Levesque's system is generalized to the statement that $\Diamond_a\varphi$ is a theorem provided that φ is a consistent formula free of modality a.

The deducibility relations of $\mathcal{HL}$ and L_I are denoted $\vdash_{\mathcal{HL}}$ and $\vdash_{L_I}$, respectively. In [12], the equivalence of L_I and $\mathcal{HL}$ was established. We will in this section prove the equivalence of the system $\AE_I$ and the system L_I. The equivalences of the three systems then follow as a corollary.

6.1 The system L_I

Let $\vdash'$ be the deducibility relation given by removing the axiom schema T from the system $\AE_I$. The deducibility relation $\vdash_{L_I}$ of the system L_I is defined as the least relation extending $\vdash'$ containing every instance of the following schema for each agent $a \in I$:

$$\Diamond_a : \Diamond_a\varphi \text{ provided } \varphi \nvdash_{L_I} \bot, \ \ \varphi \text{ free of modality } a.$$

There is a circular pattern to the $\Diamond_a$-axiom, but in [12], it is shown that the circularity is not vicious. This result is captured by Lemma 12 below.

As in $\AE_I$ any formula is provably equivalent to a first-order formula in L_I. Moreover, T is a theorem of L_I. Hence, L_I is an extension of $\AE_I$. For proof of these claims consult [13].

6.2 Equivalence of $\AE_I$ and L_I

L_I is a proper extension of $\AE_I$. However, equivalence between the systems can be established for sublanguages up to a given depth by strengthening $\AE_I$ with a particular set of formulae. In the single-agent case, when the maximal logical space is added to the axioms of Æ, the system Æ is equivalent to the propositional fragment of Levesque's system. What we need to do in the multi-agent case is to identify a set of formulae that, when added to the axioms of $\AE_I$, yields equivalence of $\AE_I$ and L_I.

Definition 3 (Maximal I-atoms). *The set of maximal I-atoms with depth k is defined as follows: Φ_0 is the set of atoms, while Φ_{k+1} is all formulae $\alpha \wedge \delta_1 \wedge \cdots \wedge \delta_n$ such that*

- *α is an atom,*
- *δ_i is a doxastic a_i-alternative spanned by $\Phi_k[a_i/\top]$,*

The critical difference between the definition of a maximal I-atom and the definition of an I-atom as defined in Definition 2 is that δ_i in the inductive step of the definition of a maximal I-atom is spanned by $\Phi_k[a_i/\top]$, and *not subsets* of $\Phi_k[a_i/\top]$. The consistency condition is furthermore omitted. This is because formulae $\alpha \wedge \delta_1 \wedge \cdots \wedge \delta_m$ trivially satisfy the consistency condition in the definition of the maximal I-atoms. (We omit the easy proof of this claim.)

The maximal logical space of agent a_i with depth k is now defined as $\lambda_i = \mathsf{Lspace}_a(\Phi_{k-1}[a_i/\top])$. We will prove that the set of formulae $\Lambda = \{\lambda_i \mid a_i \in I\}$ added to the axioms of $Æ_I$ yields equivalence with L_I up to depth k.

Before we proceed, we need an important result from [12]. This result states that L_I-consistency of a formula φ free of modality a is established without reference to the theorem $\Diamond_a \varphi$.

Lemma 12. *Let φ be L_I-provable. Then there is an L_I-proof π of φ such that $\mathsf{d}(\psi) < \mathsf{d}(\varphi)$ for every instance of an axiom $\Diamond_a \psi$ which is used in π.*

Theorem 4. *Let Λ be the set of maximal logical spaces with depth k for each agent $a_i \in I$ and $\mathsf{d}(\varphi) \leq k$. Then $\Lambda \vdash \varphi$ iff $\vdash_{L_I} \varphi$.*

Proof. The proof is by induction on the depth of the logical spaces, and both directions are proved simultaneously. As $\vdash \subseteq \vdash_{L_I}$, we need for the 'only if' direction to prove that $\vdash_{L_I} \bigwedge \Lambda$. For the 'if' direction, we need to prove that L_I is a strengthening of $Æ_I$ by $\bigwedge \Lambda$ only. That is, we need to prove that $\Diamond_{a_i} \varphi$ is deducible in $Æ_I$ from Λ, where $\Diamond_{a_i} \varphi$ is derivable in L_I by an application of $\Diamond_{a_i}$ to a formula φ, where $\mathsf{d}(\varphi) < \mathsf{d}(\lambda_i)$.

The base case is when each λ_i is spanned by the set of atoms Φ_0. 'Only if': As every atom α is L_I-consistent, $\vdash_{L_I} \Diamond_{a_i} \alpha$ by the $\Diamond_{a_i}$-axiom, and since $\bigvee \Phi_0$ is a PL-tautology, we get $\vdash_{L_I} \Box_{a_i}(\bigvee \Phi_0)$ by RN. So $\vdash_{L_I} \lambda_i$ for every $\lambda_i \in \Lambda$. 'If': Suppose $\vdash_{L_I} \Diamond_{a_i} \varphi$ is deduced in L_I by an application of $\Diamond_{a_i}$. It must then be the case that φ is a purely Boolean formula such that $\varphi \nvdash_{L_I} \bot$. Since L_I extends $Æ_I$, $\varphi \nvdash \bot$. There is then an atom α such that $\alpha \vdash \varphi$. By modal logic, $\Diamond_{a_i} \alpha \vdash \Diamond_{a_i} \varphi$, and so $\lambda_i \vdash \Diamond_{a_i} \varphi$.

In the inductive step, let $\mathsf{d}(\lambda_i) = k+1$, λ_i spanned by $\Phi_k[a_i/\top]$. 'Only if': We need to establish that $\psi \nvdash_{L_I} \bot$ for every $\psi \in \Phi_k[a_i/\top]$ and that $\vdash_{L_I} \bigvee \Phi_k[a_i/\top]$. Once these two properties are established, we may apply $\Diamond_{a_i}$ to the first and RN to the latter to get the desired result.

Note that ψ is a conjunction of an atom and a doxastic a_j-alternative δ_j for each $a_j \neq a_i$. Each δ_j entails the maximal logical space λ'_j, $\mathsf{d}(\delta_j) = k$. Let Λ' be the set of maximal logical spaces with depth k for each $a_j \neq a_i$. By construction of the logical space, we have $\psi \wedge \Lambda' \nvdash \bot$. By the induction hypothesis, $\psi \nvdash_{L_I} \bot$. By axiom $\Diamond_{a_i}$, we get $\vdash_{L_I} \Diamond_{a_i} \psi$.

Let Δ_j be the doxastic a_j-alternatives spanned by $\Phi_{k-1}[a_j/\top]$. Observe that for each $\delta_j \in \Delta_j$, $\mathsf{d}(\delta_j) = k$ and $\delta_j \vdash \lambda'_j$, where λ'_j is the maximal logical space with depth k for agent a_j. Notice that the set of conjunctions of an atom and a formula $\delta_j \in \Delta_j$ for each $a_j \neq a_i$ is exactly the set of formulae $\Phi_k[a_j/\top]$. In order to prove $\vdash_{L_I} \bigvee \Phi_k[a_i/\top]$, we will prove that $\vdash_{L_I} \bigvee \Delta_j$ for each $a_j \neq a_i$.

The result then follows by standard propositional reasoning and the fact that $\vdash_{L_I} \bigvee \Phi_0$.

We will first prove that $\lambda'_j \vdash \bigvee \Delta_j$. Suppose that $\lambda'_j \nvdash \bigvee \Delta_j$, i.e. $\lambda'_j \wedge \neg(\bigvee \Delta_j) \nvdash \bot$. By Lemma 11, there is a doxastic a_j-alternative δ'_j with depth k such that $\delta'_j \vdash \lambda'_j \wedge \neg(\bigvee \Delta_j)$. But then $\delta'_j \vdash \lambda'_j$, and so $\delta_j \in \Delta_j$. Contradiction. Since $\lambda'_j \vdash \bigvee \Delta_j$, we get $\vdash_{L_I} \bigvee \Delta_j$ by the induction hypothesis. $\vdash_{L_I} \bigvee \Phi_k[a_i/\top]$ follows by standard propositional reasoning, and $\vdash_{L_I} \Box_{a_i}(\bigvee \Phi_k[a_i/\top])$ by RN.

'If': Suppose $\vdash_{L_I} \lozenge_{a_i}\varphi$, $\mathsf{d}(\varphi) < \mathsf{d}(\lambda_i)$, $\lambda_i \in \Lambda$, is deduced in L_I by an application of $\lozenge_{a_i}$. It must then be the case that φ is a formula free of modality a_i such that $\varphi \nvdash_{L_I} \bot$. By Lemma 12, any application of the $\lozenge_{a_i}$-axiom to establish the consistency of φ is to formulae with depth $< \mathsf{d}(\varphi)$. By the induction hypothesis, $\Lambda' \wedge \varphi \nvdash \bot$, where Λ' is the set of maximal logical spaces with depth k for each $a_j \neq a_i$.

We may without loss of generality assume that φ is first-order and on DNF. Since $\Lambda' \wedge \varphi \nvdash \bot$, there is a disjunct ψ of ϕ such that $\Lambda' \wedge \psi \nvdash \bot$. ψ is a conjunction of a purely Boolean formula $\psi^{\mathcal{P}}$ and a completely a_j-modalized formula ψ^{a_j} for each $a_j \neq a_i$. Since $\lambda'_j \wedge \psi^{a_j} \nvdash \bot$, $\lambda'_j \in \Lambda'$, there is by Lemma 11 a doxastic a_j-alternative δ_j with depth k such that $\delta_j \vdash \lambda'_j \wedge \psi^{a_j}$. Let Δ be the set of these formulae δ_j for each $a_j \neq a_i$. As to $\psi^{\mathcal{P}}$, there is an atom α such that $\alpha \vdash \psi^{\mathcal{P}}$. Since each δ_j entails the maximal logical space, the consistency condition is trivially satisfied, and so $\alpha \wedge \Delta \nvdash \bot$. Since each element of $\{\alpha\} \cup \Delta$ entails a respective conjunct of ψ, we have $\alpha \wedge \Delta \vdash \psi$, and so $\alpha \wedge \Delta \vdash \varphi$. Observe that the conjunction $\alpha \wedge \bigwedge \Delta$ is an element of $\Phi_k[a_j/\top]$ and that $\lozenge_{a_i}(\alpha \wedge \bigwedge \Delta)$ is a conjunct of the maximal logical space λ_i with depth k. Since $\alpha \wedge \Delta \vdash \varphi$, we have $\lozenge_{a_i}(\alpha \wedge \bigwedge \Delta) \vdash \lozenge_{a_i}\varphi$ by modal logic, and so $\lambda_i \vdash \lozenge_{a_i}\varphi$ as desired. $\square$

Corollary 3. *$\Lambda \vdash \varphi$ iff $\vdash_{L_I} \varphi$ iff $\vdash_{\mathit{HL}} \varphi$, $\varphi \in \mathcal{L}$, provided $\mathsf{d}(\varphi) \leq \mathsf{d}(\lambda_i)$ for each $\lambda_i \in \Lambda$.*

Proof. Follows immediately from Theorem 15 of [12] and Theorem 4. $\square$

7 Conclusion and Future Work

The focus of this paper is on the logical foundation of multi-agent systems. We have successfully developed a notion of logical space for agents in a multi-modal only knowing language. Clearly, a practical application will require a more economical way of representing and reasoning within logical spaces, typically achieved by means of highly restricted languages. However, to implement constraints like this, one needs to know what "all the options" are. This paper presents an answer to this fundamental and conceptually important question.

A number of interesting questions can be raised on the basis of this logical clarification. First, we have not presented any complexity analysis. The size of a logical space grows quickly beyond any tractable level. However, in a particular situation one will not need to span the entire space syntactically, exactly like one in Æ can provide an implicit definition of a logical space by means of a characteristic formula [8, 11]. We plan to address this question in a subsequent paper. We also plan to extend the reduction method used to give a constructive proof of the Modal Reduction Theorem in Æ to $Æ_I$ and to extend the language with language constructs to express different degrees of confidence for each agent (like in Æ). The latter task is in itself straightforward; however, a non-trivial use of this would be to develop a theory of multi-agent default reasoning within this language which generalizes the encoding of default logic in Æ [1].

References

1. Engan, I., Lian, E. H., Langholm, T. and Waaler, A.: Default Reasoning with Preference within Only Knowing Logic. To appear in Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05), Springer Verlag Lecture Notes in Artificial Intelligence series (2005)
2. Halpern, J. Y.: Reasoning about Only Knowing with Many Agents. Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-93) (1993)
3. Halpern, J. Y.: A Theory of Knowledge and Ignorance for many agents. Journal of Logic and Computation **7:1** (1997) 79–108
4. Halpern, J. Y. and Lakemeyer, G.: Multi-Agent Only Knowing. Journal of Logic and Computation **11:1** (2001) 40–70
5. Hoek, W. and Thijsse, E.: A General Approach to Multi-Agent Minimal Knowledge: With Tools and Samples. Studia Logica **72:1** (2002) 61–84
6. Lakemeyer, G.: All They Know: A Study in Multi-Agent Autoepistemic Reasoning. Proc. of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93) (1993) 376–381
7. Levesque, H. J.: All I know: A study in autoepistemic logic. Artificial Intelligence **42** (1990) 263–309
8. Lian, E. H., Angle, T. and Waaler, A.: Only Knowing with Confidence Levels: Reductions and Complexity. JELIA 2004, Proceedings, Lecture Notes in Computer Science **3229** (2004) 500–512
9. Solhaug, B.: Logical Spaces in Multi-Modal Only Knowing Logics. Master's Thesis, University of Oslo (2004)
10. Waaler, A.: Logical Studies in Complementary Weak S5. Doctoral thesis, University of Oslo (1994)
11. Waaler, A., Klüwer, J.W., Langholm, T. and Lian, E.: Only Knowing with Degrees of Confidence. Submitted for publication, 2005.
12. Waaler, A.: Consistency proofs for systems of multi-agent only knowing. To appear in Advances in Modal Logic 2005.
13. Waaler, A. and Solhaug, B.: Semantics for Multi-Agent Only Knowing (extended abstract). To appear in the Proceedings of Theoretical Aspects of Rationality and Knowledge (TARK X), 2005.

Combining Answer Sets of Nonmonotonic Logic Programs

Chiaki Sakama[1] and Katsumi Inoue[2]

[1] Department of Computer and Communication Sciences
Wakayama University, Sakaedani, Wakayama 640-8510, Japan
`sakama@sys.wakayama-u.ac.jp`
[2] National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
`ki@nii.ac.jp`

Abstract. This paper studies compositional semantics of nonmonotonic logic programs. We suppose the answer set semantics of extended disjunctive programs and consider the following problem. Given two programs P_1 and P_2, which have the sets of answer sets $\mathcal{AS}(P_1)$ and $\mathcal{AS}(P_2)$, respectively; find a program Q which has answer sets as minimal sets $S \cup T$ for S from $\mathcal{AS}(P_1)$ and T from $\mathcal{AS}(P_2)$. The program Q combines answer sets of P_1 and P_2, and provides a compositional semantics of two programs. Such program composition has application to coordinating knowledge bases in multi-agent environments. We provide methods for computing program composition and discuss their properties.

1 Introduction

Combining knowledge of different information sources is a central topic in multi-agent systems. In those environments, different agents generally have different knowledge and belief, then coordination among agents is necessary to form acceptable agreements. In computational logic, knowledge and belief of an agent are represented by a set of formulas. Combining multiple knowledge bases is then formulated as the problem of composing different theories. In multi-agent environments, individual agents are supposed to have incomplete information. Since theories including incomplete information are *nonmonotonic*, it is important and meaningful to develop a framework of composing nonmonotonic theories.

To see the problem, suppose the following scenario: there is a trouble in a system which consists of three components c_1, c_2, and c_3. After some diagnoses, an expert e_1 concludes that the trouble would be caused by one of the two components c_1 and c_2, but they are unlikely to be in trouble at the same time. On the other hand, another expert e_2 concludes that the trouble would be caused by one of the two components c_2 and c_3, while they would not disorder simultaneously. Two experts' diagnoses are encoded as the following logic programs:

$$\begin{aligned} e_1 :\ & c_1 \leftarrow not\, c_2, \\ & c_2 \leftarrow not\, c_1, \end{aligned}$$

$$\begin{aligned} e_2:\ & c_2 \leftarrow \mathit{not}\, c_3, \\ & c_3 \leftarrow \mathit{not}\, c_2. \end{aligned}$$

Here, *not* represents *negation as failure* and the rules $c_i \leftarrow \mathit{not}\, c_j$ and $c_j \leftarrow \mathit{not}\, c_i$ encode two alternative causes. By merging two programs, the program $e_1 \cup e_2$ has two *answer sets* $\{c_1, c_3\}$ and $\{c_2\}$, which would be acceptable to each expert. (Note: e_1 (resp. e_2) represents that c_1 and c_2 (resp. c_2 and c_3) are alternative causes of the problem, but each expert does not exclude the possibility of having c_1 and c_3 at the same time.)

The story goes on: e_1 consider that the possible cause is either c_1 or c_2, but he empirically knows that c_1 is more likely to cause the trouble. Similarly, e_2 consider that the possible cause is either c_2 or c_3, but she empirically knows that c_2 is more likely to cause the trouble. Two experts then slightly modify their diagnoses as

$$\begin{aligned} e_1':\ & c_1 \leftarrow \mathit{not}\, c_2, \\ & c_2 \leftarrow \neg\, c_1, \\ e_2':\ & c_2 \leftarrow \mathit{not}\, c_3, \\ & c_3 \leftarrow \neg\, c_2. \end{aligned}$$

After the modification, e_1' is read as: c_1 is considered a cause if there is no evidence of c_2, and c_2 will not become a cause unless c_1 is explicitly negated. e_2' is read in a similar way. Merging two programs, however, the program $e_1' \cup e_2'$ has the single answer set $\{c_2\}$, which reflects the result of diagnosis by e_2' but does not reflect e_1'. When two experts are equally reliable, the result might be unsatisfactory. In fact, e_2' puts weight on c_2 relative to c_3 and e_1' puts weight on c_1 relative to c_2. After integrating these diagnoses, there is no reason to conclude c_2 as the plausible conclusion.

The above example illustrates that composition of nonmonotonic theories is not achieved by simply merging them. The problem is then how to build a compositional semantics of nonmonotonic theories. In this paper, we consider composition of *extended disjunctive programs* under the *answer set semantics* [11]. An answer set is a set of literals which corresponds to a belief set being built by a rational reasoner on the basis of a program [2]. A program generally has multiple answer sets, and different agents have different collections of answer sets. We then capture composition of two programs as the problem of building a new program which combines answer sets of the original programs. Formally, the problems considered in this paper are described as follows.

Given: two programs P_1 and P_2;

Find: a program Q satisfying $\mathcal{AS}(Q) = \mathit{min}(\mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2))$ where $\mathcal{AS}(P)$ represents the set of answer sets of a program P and $\mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2) = \{\, S \cup T \mid S \in \mathcal{AS}(P_1) \text{ and } T \in \mathcal{AS}(P_2)\}$,

where $\mathit{min}(X) = \{\, Y \in X \mid \neg\exists Z \in X \text{ s.t. } Z \subset Y\,\}$. The program Q satisfying the above condition is called *composition* of P_1 and P_2. The result of composition combines answer sets of two programs, which has the effect of amalgamating the original belief of each agent. We develop methods for constructing a program having the compositional semantics.

The rest of this paper is organized as follows. Section 2 introduces basic notions used in this paper. Section 3 presents compositional semantics and its technical properties. Section 4 provides methods for building programs which reflect compositional semantics. Section 5 addresses permissible composition for multi-agent coordination. Section 6 discusses related issues and Section 7 summarizes the paper.

2 Preliminaries

In this paper, we suppose an agent that has a knowledge base written in logic programming.

A *program* considered in this paper is an *extended disjunctive program* (EDP) which is a set of *rules* of the form:

$$L_1 \,;\, \cdots \,;\, L_l \;\leftarrow\; L_{l+1}\,, \ldots,\, L_m,\, not\, L_{m+1}\,, \ldots,\, not\, L_n \qquad (n \geq m \geq l \geq 0)$$

where each L_i is a positive/negative literal, i.e., A or $\neg A$ for an atom A, and *not* is *negation as failure* (NAF). $not\, L$ is called an *NAF-literal*. The symbol ";" represents disjunction. The left-hand side of the rule is the *head*, and the right-hand side is the *body*. For each rule r of the above form, $head(r)$, $body^+(r)$ and $body^-(r)$ denote the sets of literals $\{L_1, \ldots, L_l\}$, $\{L_{l+1}, \ldots, L_m\}$, and $\{L_{m+1}, \ldots, L_n\}$, respectively. Also, $not\_body^-(r)$ denotes the set of NAF-literals $\{not\, L_{m+1}, \ldots, not\, L_n\}$. A disjunction of literals and a conjunction of (NAF-)literals in a rule are identified with its corresponding sets of literals. A rule r is often written as $head(r) \leftarrow body^+(r), not\_body^-(r)$ or $head(r) \leftarrow body(r)$ where $body(r) = body^+(r) \cup not\_body^-(r)$. A rule r is *disjunctive* if $head(r)$ contains more than one literal. A rule r is an *integrity constraint* if $head(r) = \emptyset$; and r is a *fact* if $body(r) = \emptyset$. A program is an *extended logic program* (ELP) if it contains no disjunctive rule. A program is *NAF-free* if no rule contains NAF-literals. A program with variables is semantically identified with its ground instantiation, and we handle propositional and ground programs only.

The semantics of EDPs is given by the *answer set semantics* [11]. Let Lit be the set of all ground literals in the language of a program. A set $S(\subset Lit)$ *satisfies* a ground rule r if $body^+(r) \subseteq S$ and $body^-(r) \cap S = \emptyset$ imply $head(r) \cap S \neq \emptyset$. In particular, S satisfies a ground integrity constraint r with $head(r) = \emptyset$ if either $body^+(r) \not\subseteq S$ or $body^-(r) \cap S \neq \emptyset$. S satisfies a ground program P if S satisfies every rule in P.

Let P be an NAF-free EDP. Then, a set $S(\subseteq Lit)$ is an *answer set* of P if S is a minimal set such that

1. S satisfies every rule from the ground instantiation of P,
2. If S contains a pair of complementary literals L and $\neg L$, $S = Lit$.

Next, let P be any EDP and $S \subseteq Lit$. For every rule r in the ground instantiation of P, the rule $head(r) \cap S \leftarrow body^+(r)$ is included in the *reduct* ${}^{S}P$ if $body^+(r) \subseteq S$ and $body^-(r) \cap S = \emptyset$. Then, S is an *answer set* of P if S is an answer set of ${}^{S}P$.

Remark: The definition of a reduct presented above is slightly different from the original one in [11]. In [11], the rule $head(r) \leftarrow body^+(r)$ is included in the reduct P^S (called Gelfond-Lifschitz reduction) if $body^-(r) \cap S = \emptyset$. Our reduction imposes additional conditions, but two reductions produce the same answer sets of EDPs.

Proposition 2.1 *For any EDP P, S is an answer set of ${}^{S}P$ iff S is an answer set of P^S.*

Proof. If S is an answer set of P^S, it is a minimal set satisfying every rule in P^S. For any rule r in ${}^{S}P \setminus P^S$, it holds $body^+(r) \subseteq S$, $(head(r) \leftarrow body^+(r)) \in P^S$ and $(head(r) \cap S \leftarrow body^+(r)) \in {}^{S}P$. As S satisfies P^S, $body^+(r) \subseteq S$ implies $head(r) \cap S \neq \emptyset$. So, S satisfies ${}^{S}P$. Assume that there is a minimal set $T \subset S$ satisfying every rule in ${}^{S}P$. Any rule r in $P^S \setminus {}^{S}P$ satisfies either (a) $body^+(r) \not\subseteq S$ or (b) $body^+(r) \subseteq S$, $(head(r) \leftarrow body^+(r)) \in P^S$ and $(head(r) \cap S \leftarrow body^+(r)) \in {}^{S}P$. In case of (a), $body^+(r) \not\subseteq S$ implies $body^+(r) \not\subseteq T$. Then, T satisfies r. In case of (b), as T satisfies ${}^{S}P$, $body^+(r) \subseteq T$ implies $T \cap (head(r) \cap S) \neq \emptyset$, thereby $T \cap head(r) \neq \emptyset$. Thus, in each case T satisfies every rule in P^S. This contradicts the fact that S is a minimal set satisfying P^S. Then, S is also a minimal set satisfying every rule in ${}^{S}P$. Hence, S is an answer set of P. The converse is shown in a similar manner. □

Example 2.1. Let P be the program:

$$\begin{aligned} &p\,;\,q \leftarrow, \\ &q \leftarrow p, \\ &r \leftarrow not\, p. \end{aligned}$$

For $S = \{q, r\}$, $\quad P^S$ becomes

$$\begin{aligned} &p\,;\,q \leftarrow, \\ &q \leftarrow p, \\ &r \leftarrow, \end{aligned}$$

while ${}^{S}P$ becomes

$$\begin{aligned} &q \leftarrow, \\ &r \leftarrow . \end{aligned}$$

Each reduct produces the same answer set S. Note that $\{p, q\}$ does not become an answer set of P.

For later convenience, we use the reduct ${}^{S}P$ for computing answer sets of P.

A program has none, one, or multiple answer sets in general. The set of all answer sets of P is written as $\mathcal{AS}(P)$. A program having a single answer set is called *categorical* [2]. Categorical programs include important classes of programs such as *definite programs*, *stratified programs*, and *call-consistent programs*. Every NAF-free ELP has a single answer set. An answer set is *consistent* if it is not *Lit*. A program P is *consistent* if it has a consistent answer set; otherwise, P is *inconsistent*.

A literal L is a consequence of *credulous reasoning* in a program P (written as $L \in crd(P)$) if L is included in some answer set of P. A literal L is a consequence of *skeptical reasoning* in a program P (written as $L \in skp(P)$) if L is included in every answer set of P. Clearly, $skp(P) \subseteq crd(P)$ for any P.

3 Combining Answer Sets

In this section, we introduce a compositional semantics of programs. Throughout the paper, different programs are assumed to have the same underlying language with a fixed interpretation.

Let S and T be two sets of literals. Then, define

$$S \uplus T = \begin{cases} S \cup T\,, & \text{if } S \cup T \text{ is consistent;} \\ Lit\,, & \text{otherwise.} \end{cases}$$

For two collections $\mathcal{S}$ and $\mathcal{T}$ of sets, define

$$\mathcal{S} \uplus \mathcal{T} = \{\, S \uplus T \mid S \in \mathcal{S} \text{ and } T \in \mathcal{T} \,\}.$$

Definition 3.1. Let P_1 and P_2 be two consistent programs. A program Q is called a *composition* of P_1 and P_2 if it satisfies the condition

$$\mathcal{AS}(Q) = min(\mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2))$$

where $min(X) = \{\, Y \in X \mid \neg\exists Z \in X \text{ s.t. } Z \subset Y \,\}$.

The set $\mathcal{AS}(Q)$ is called the *compositional semantics* of P_1 and P_2. By the definition, the compositional semantics is defined as the collection of minimal sets which are obtained by combining answer sets of the original programs.

Example 3.1. Let $\mathcal{AS}(P_1) = \{\{p\}, \{q\}\}$ and $\mathcal{AS}(P_2) = \{\{p\}, \{r\}\}$. Then, the compositional semantics becomes $\mathcal{AS}(Q) = \{\, \{p\}, \{q, r\} \,\}$.

Note that we do not consider composition of inconsistent programs, because such composition appears meaningless and trivial. So in program composition consistent programs are handled hereafter.

Proposition 3.1 *Let P_1 and P_2 be two consistent programs, and Q a result of composition. Then, $\forall S \in \mathcal{AS}(Q)$, $\exists T \in \mathcal{AS}(P_1) \cup \mathcal{AS}(P_2)$ s.t. $T \subseteq S$.*

Proposition 3.1 presents that every answer set in the compositional semantics extends some answer sets of the original programs. On the other hand, the original programs may have an answer set which does not have its extension in their compositional semantics.

Example 3.2. Let $\mathcal{AS}(P_1) = \{\{p, q\}\}$ and $\mathcal{AS}(P_2) = \{\{p\}, \{q, r\}\}$. The compositional semantics of P_1 and P_2 becomes $\mathcal{AS}(Q) = \{\{p, q\}\}$ which extends $\{p, q\}$ of P_1 and $\{p\}$ of P_2, but does not extend $\{q, r\}$ of P_2.

In the above example, $\{p, q\}$ absorbs $\{p\}$ and remains as a result of composition. Consequently, the set $\{p, q, r\}$, which combines $\{p, q\}$ of P_1 and $\{q, r\}$ of P_2, becomes non-minimal and is excluded from the result of composition.

Such cases are formally stated as follows.

Definition 3.2. Let P_1 and P_2 be two consistent programs, and Q a result of composition. When $\mathcal{AS}(Q) = \mathcal{AS}(P_1)$, P_1 *absorbs* P_2.

In Example 3.2, P_1 absorbs P_2. If one program absorbs another program, the compositional semantics coincides with one of the original programs. The next proposition characterizes situations in which absorption happens.

Proposition 3.2 *Let P_1 and P_2 be two consistent programs, and Q a result of composition. Then, P_1 absorbs P_2 iff for any $S \in \mathcal{AS}(P_1)$, there is $T \in \mathcal{AS}(P_2)$ such that $T \subseteq S$.*

Proof. For any $S \in \mathcal{AS}(P_1)$, suppose that there is $T \in \mathcal{AS}(P_2)$ such that $T \subseteq S$. As $S \cup T = S$, $\mathcal{AS}(P_1) \subseteq \mathcal{AS}(Q)$. Suppose any $T' \in \mathcal{AS}(P_2)$ such that $T' \not\subseteq S$ for any $S \in \mathcal{AS}(P_1)$. Then, $S \subset S \cup T'$. Since $S \in \mathcal{AS}(Q)$, $S \cup T' \notin \mathcal{AS}(Q)$. Thus, $\mathcal{AS}(Q) \setminus \mathcal{AS}(P_1) = \emptyset$. Hence, $\mathcal{AS}(Q) = \mathcal{AS}(P_1)$.

Conversely, if $\mathcal{AS}(Q) = \mathcal{AS}(P_1)$, for any $S \in \mathcal{AS}(P_1)$ there is $T \in \mathcal{AS}(P_2)$ such that $S = S \cup T$. Then, $T \subseteq S$. □

Skeptical/credulous inference in compositional semantics has the following properties.

Proposition 3.3 *Let P_1 and P_2 be two consistent programs, and Q a result of composition. Then,*

1. *$crd(Q) \subseteq crd(P_1) \cup crd(P_2)$.*
2. *$skp(Q) = skp(P_1) \cup skp(P_2)$.*

Proof. The result of (1) holds by Proposition 3.1. To see (2), if any literal L is included in every answer set S in $\mathcal{AS}(P_1)$ or included in every answer set T in $\mathcal{AS}(P_2)$, it is included in every $S \cup T$ in $\mathcal{AS}(Q)$. Conversely, if any literal L is included in every answer set U in $\mathcal{AS}(Q)$, L is included in every minimal set $S \cup T$ for some $S \in \mathcal{AS}(P_1)$ and $T \in \mathcal{AS}(P_2)$. Suppose $L \in S$ and there is $S' \in \mathcal{AS}(P_1)$ such that $L \notin S'$. Then, there is $T \in \mathcal{AS}(P_2)$ such that $L \in T$. If there is $T' \in \mathcal{AS}(P_2)$ such that $L \notin T'$, then $L \notin S' \cup T'$ thereby there is $V \in \mathcal{AS}(Q)$ such that $L \notin V$. Contradiction. □

Example 3.3. Let $\mathcal{AS}(P_1) = \{\{p, q\}\}$ and $\mathcal{AS}(P_2) = \{\{p\}, \{q, r\}\}$ where $crd(P_1) = skp(P_1) = \{ p, q \}$, $crd(P_2) = \{ p, q, r\}$, and $skp(P_2) = \emptyset$. The compositional semantics of P_1 and P_2 becomes $\mathcal{AS}(Q) = \{\{p, q\}\}$ where $crd(Q) = skp(Q) = \{p, q\}$.

The result of composition possibly becomes inconsistent even if the original programs are consistent.

Example 3.4. Let $\mathcal{AS}(P_1) = \{\{p\}\}$ and $\mathcal{AS}(P_2) = \{\{\neg p\}\}$. Then, $\mathcal{AS}(Q) = \{ Lit \}$.

When $\mathcal{AS}(Q)$ has no consistent answer set, we consider that program composition fails. A necessary and sufficient condition to have a successful program composition is as follows.

Proposition 3.4 *Let P_1 and P_2 be consistent programs, and Q a result of composition. Then, Q is consistent iff there are $S \in \mathcal{AS}(P_1)$ and $T \in \mathcal{AS}(P_2)$ such that $S \cup T$ is consistent.*

Proof. Q is consistent iff there is a consistent set $S \cup T$ in $\mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2)$ for $S \in \mathcal{AS}(P_1)$ and $T \in \mathcal{AS}(P_2)$. Hence, the result follows. □

In program composition, the problem of interest is the cases where one program does not absorb the other and the result of composition is consistent. In the next section, we present methods for computing program composition.

4 Composing Programs

In this section, every program is supposed to have a finite number of answer sets. We first introduce an additional notation used in this section. Given programs $P_1, \ldots, P_k$, define

$$P_1 \,;\, \cdots \,;\, P_k = \\ \{\, head(r_1); \cdots ; head(r_k) \leftarrow body(r_1), \ldots, body(r_k) \;\mid\; r_i \in P_i \ (1 \le i \le k) \,\}.$$

Definition 4.1. Given two programs P_1 and P_2,

1. Compute $R(S, T) = {}^{S}P_1 \cup {}^{T}P_2$ for every $S \in \mathcal{AS}(P_1)$ and $T \in \mathcal{AS}(P_2)$.
2. Let $\mathcal{AS}(P_1) = \{ S_1, \ldots, S_m \}$ and $\mathcal{AS}(P_2) = \{ T_1, \ldots, T_n \}$. Then, define

$$P_1 \odot P_2 = R(S_1, T_1) \,;\, \cdots \,;\, R(S_m, T_n)$$

   where $R(S_1, T_1), \ldots, R(S_m, T_n)$ is any enumeration of the $R(S, T)$'s constructed in Step 1.

By the definition, $P_1 \odot P_2$ is computed in time $|P_1| \times |P_2| \times |\mathcal{AS}(P_1)| \times |\mathcal{AS}(P_2)|$, where $|P|$ represents the number of rules in P and $|\mathcal{AS}(P)|$ represents the number of answer sets of P. In particular, if P_1 and P_2 respectively have the single answer set $\mathcal{AS}(P_1) = \{S\}$ and $\mathcal{AS}(P_2) = \{T\}$, it becomes $P_1 \odot P_2 = {}^{S}P_1 \cup {}^{T}P_2$.

The program $P_1 \odot P_2$ generally contains useless or redundant literals/rules, and the following program transformations are useful to simplify the program: (i) Delete a rule r from a program if $\mathit{head}(r) \cap \mathit{body}^{+}(r) \neq \emptyset$ (*elimination of tautologies*: TAUT); (ii) Delete a rule r from a program if there is another rule r' in the program such that $\mathit{head}(r') \subseteq \mathit{head}(r)$ and $\mathit{body}(r') \subseteq \mathit{body}(r)$ (*elimination of non-minimal rules*: NONMIN); (iii) A disjunction $(L; L)$ appearing in $\mathit{head}(r)$ is merged into L, and a conjunction (L, L) appearing in $\mathit{body}(r)$ is merged into L (*merging duplicated literals*: DUPL). These program transformations all preserve the answer sets of an EDP [4].

Example 4.1. Consider two programs:

$$\begin{aligned} P_1 :\ & p \leftarrow \mathit{not}\, q, \\ & q \leftarrow \mathit{not}\, p, \\ & s \leftarrow p, \\ P_2 :\ & p \leftarrow \mathit{not}\, r, \\ & r \leftarrow \mathit{not}\, p, \end{aligned}$$

where $\mathcal{AS}(P_1) = \{\{p, s\}, \{q\}\}$ and $\mathcal{AS}(P_2) = \{\{p\}, \{r\}\}$. Then, there are four $R(S, T)$ such that

$$\begin{aligned} R(\{p, s\}, \{p\}) :\ & p \leftarrow, \quad s \leftarrow p, \\ R(\{p, s\}, \{r\}) :\ & p \leftarrow, \quad s \leftarrow p, \quad r \leftarrow, \\ R(\{q\}, \{p\}) :\ & q \leftarrow, \quad p \leftarrow, \\ R(\{q\}, \{r\}) :\ & q \leftarrow, \quad r \leftarrow . \end{aligned}$$

$P_1 \odot P_2$ contains the following seven rules (after applying DUPL):

$$\begin{aligned} & p\, ;\, q \leftarrow, \\ & p\, ;\, r \leftarrow, \\ & p\, ;\, q\, ;\, r \leftarrow, \\ & q\, ;\, s \leftarrow p, \\ & q\, ;\, r\, ;\, s \leftarrow p, \\ & p\, ;\, q\, ;\, s \leftarrow p, \\ & p\, ;\, r\, ;\, s \leftarrow p. \end{aligned}$$

Further, those rules, other than the first one, the second one, and the fourth one, are eliminated by NONMIN. Consequently, the simplified program becomes

$$\begin{aligned} & p\, ;\, q \leftarrow, \\ & p\, ;\, r \leftarrow, \\ & q\, ;\, s \leftarrow p. \end{aligned}$$

The operator $\odot$ has the following properties.

Proposition 4.1 *The operation $\odot$ is commutative and associative.*

Proof. The commutative law $P_1 \odot P_2 = P_2 \odot P_1$ is straightforward. To see the associative law, both $(P_1 \odot P_2) \odot P_3$ and $P_1 \odot (P_2 \odot P_3)$ consist of rules of the form: $head(r_1)\,;\, \cdots\,;\, head(r_k) \leftarrow body(r_1), \ldots, body(r_k)$ for $r_i \in R(S,T,U)$ $(1 \leq i \leq k)$ where $R(S,T,U) = {}^{S}P_1 \cup {}^{T}P_2 \cup {}^{U}P_3$ for any $S \in \mathcal{AS}(P_1)$, $T \in \mathcal{AS}(P_2)$, and $U \in \mathcal{AS}(P_3)$. Hence, $(P_1 \odot P_2) \odot P_3 = P_1 \odot (P_2 \odot P_3)$. $\square$

Now we proceed to show the main result of this paper.

Lemma 4.2 *Let P_1 and P_2 be two consistent programs, and $S \in \mathcal{AS}(P_1)$ and $T \in \mathcal{AS}(P_2)$. Then, $S \cup T$ is an answer set of ${}^{S}P_1 \cup {}^{T}P_2$.*

Proof. S is a minimal set satisfying ${}^{S}P_1$ and T is a minimal set satisfying ${}^{T}P_2$. Since $body(r) \subseteq S$ and $head(r) \subseteq S$ for any $r \in {}^{S}P_1$ and $body(r') \subseteq T$ and $head(r') \subseteq T$ for any $r' \in {}^{T}P_2$, $S \cup T$ satisfies ${}^{S}P_1 \cup {}^{T}P_2$. Suppose that there is $T' \subset T$ such that $S \cup T'$ satisfies ${}^{S}P_1 \cup {}^{T}P_2$. For any $L \in T \setminus T'$, if $L \notin S$, T' satisfies ${}^{T}P_2$. But this cannot happen, since T is a minimal set satisfying ${}^{T}P_2$. Then, $L \in S$, thereby $S \cup T = S \cup T'$. Thus, $S \cup T$ is a minimal set satisfying ${}^{S}P_1 \cup {}^{T}P_2$. As ${}^{S}P_1 \cup {}^{T}P_2$ is NAF-free, $S \cup T$ becomes an answer set of it. $\square$

Lemma 4.3 *If U is a minimal set satisfying $(R(S,T)\,;\, R(S',T'))$, U is a minimal set satisfying $R(S,T)$.*

Proof. If there is $V \subset U$ satisfying $R(S,T)$, for any rule $r \in R(S,T)$ it holds $body(r) \not\subseteq V$ or $head(r) \subseteq V$. Then, V satisfies every rule $head(r); head(r') \leftarrow body(r), body(r')$ in $(R(S,T)\,;\, R(S',T'))$ for any $r' \in R(S',T')$. This contradicts the fact that U is a minimal set satisfying $(R(S,T)\,;\, R(S',T'))$. $\square$

Theorem 4.4. *Let P_1 and P_2 be two consistent programs. Then, $\mathcal{AS}(P_1 \odot P_2) = min(\mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2))$.*

Proof. Let $U \in min(\mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2))$. Then, there is $S \in \mathcal{AS}(P_1)$ and $T \in \mathcal{AS}(P_2)$ such that $U = S \cup T$. By Lemma 4.2, U is an answer set of $R(S,T)$. Then, U satisfies $P_1 \odot P_2$. Suppose that there is a minimal set $V \subset U$ which satisfies $P_1 \odot P_2$. In this case, V is a minimal set satisfying some $R(S',T')$ in $P_1 \odot P_2$ (Lemma 4.3). It then holds that $V = S' \cup T'$ for some $S' \in \mathcal{AS}(P_1)$ and $T' \in \mathcal{AS}(P_2)$ (by Lemma 4.2). Since $V \in \mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2)$ and $V \subset U$, $U \notin min(\mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2))$. Contradiction. Thus, U is a minimal set satisfying $P_1 \odot P_2$, so $U \in \mathcal{AS}(P_1 \odot P_2)$.

Conversely, let $U \in \mathcal{AS}(P_1 \odot P_2)$. Then, U is a minimal set satisfying some $R(S,T)$ in $P_1 \odot P_2$ (Lemma 4.3). It then holds $U = S \cup T$ for some $S \in \mathcal{AS}(P_1)$ and $T \in \mathcal{AS}(P_2)$ (by Lemma 4.2). Thus, $U \in \mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2)$. Suppose that there is a minimal set $V \subset U$ such that $V = S' \cup T'$ for some $S' \in \mathcal{AS}(P_1)$ and $T' \in \mathcal{AS}(P_2)$. In this case, $V \in min(\mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2))$, and V becomes an answer set of $P_1 \odot P_2$ by the proof presented above. This contradicts the assumption of $U \in \mathcal{AS}(P_1 \odot P_2)$. Hence, $U \in min(\mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2))$. $\square$

Example 4.2. In Example 4.1, $\mathcal{AS}(P_1 \odot P_2) = \{\{p, q\}, \{p, s\}, \{q, r\}\}$, which coincides with the result of composition.

Two programs P_1 and P_2 are *merged* by taking their union $P_1 \cup P_2$. Program composition and merging bring syntactically and semantically different results in general, but there are some relations for special cases.

Proposition 4.5 *For two consistent NAF-free programs P_1 and P_2, if $P_1 \cup P_2$ is consistent, $P_1 \odot P_2$ is consistent.*

Proof. If $P_1 \cup P_2$ is consistent, there is ${}^{S}P_1$ for $S \in \mathcal{AS}(P_1)$ and ${}^{T}P_2$ for $T \in \mathcal{AS}(P_2)$ such that ${}^{S}P_1 \cup {}^{T}P_2$ is consistent. Then, $S \cup T$ is consistent. By Proposition 3.4 and Theorem 4.4, $P_1 \odot P_2$ is consistent. □

The converse of Proposition 4.5 does not hold in general.

Example 4.3. Let $P_1 = \{\, p \leftarrow \,\}$ and $P_2 = \{\, \leftarrow p \,\}$. Then, $P_1 \odot P_2 = \{\, p \leftarrow \,\}$, but $P_1 \cup P_2$ has no answer set.

In the general case, there is no relation for the "easiness" of inconsistency arising between composition and merging.

Example 4.4. Let $P_1 = \{\, p \leftarrow \mathit{not}\, \neg p \,\}$ and $P_2 = \{\, \neg p \leftarrow \mathit{not}\, p \,\}$. Then, $P_1 \cup P_2$ is consistent, but $P_1 \odot P_2 = \{\, p \leftarrow\ ,\ \ \neg p \leftarrow \,\}$ is inconsistent. On the other hand, let $P_3 = \{\, p \leftarrow \mathit{not}\, q,\ \ q \leftarrow \mathit{not}\, r \,\}$ and $P_4 = \{\, r \leftarrow \mathit{not}\, p \,\}$. Then, $P_3 \cup P_4$ is inconsistent, but $P_3 \odot P_4 = \{\, q \,;\, r \leftarrow \,\}$ is consistent.

For extended logic programs, the following syntactical and semantical relations hold.

Proposition 4.6 *For two consistent NAF-free ELPs P_1 and P_2, $P_1 \odot P_2 \subseteq P_1 \cup P_2$.*

Proof. In this case, each program has the single answer set. Let $\mathcal{AS}(P_1) = \{S\}$ and $\mathcal{AS}(P_2) = \{T\}$. Then, $P_1 \setminus {}^{S}P_1 = \{\, r \mid r \in P_1 \text{ and } \mathit{body}(r) \not\subseteq S \,\}$, and ${}^{S}P_1 \setminus P_1 = \emptyset$. This is also the case for P_2. Since $P_1 \odot P_2 = {}^{S}P_1 \cup {}^{T}P_2$, the result follows. □

Proposition 4.7 *Let P_1 and P_2 be two consistent NAF-free ELPs. Then, $U \subseteq V$ holds for the answer set U of $P_1 \odot P_2$ and the answer set V of $P_1 \cup P_2$.*

Proof. Let $\mathcal{AS}(P_1) = \{S\}$ and $\mathcal{AS}(P_2) = \{T\}$. Then, $\mathcal{AS}(P_1 \odot P_2) = \{S \cup T\}$. On the other hand, if $P_1 \cup P_2$ is inconsistent, $\mathcal{AS}(P_1 \cup P_2) = \{\mathit{Lit}\}$. So, $S \cup T \subseteq \mathit{Lit}$. Else if $P_1 \cup P_2$ has the consistent answer set V, $S \cup T$ is consistent by Proposition 4.5. Then, $S \cup T \subset V$ by Proposition 4.6. □

Example 4.5. Let $P_1 = \{\, p \leftarrow q \,\}$ and $P_2 = \{\, q \leftarrow \,\}$. Then, $P_1 \odot P_2 = \{\, q \leftarrow \,\}$ and $P_1 \cup P_2 = \{\, p \leftarrow q,\ \ q \leftarrow \,\}$. So $P_1 \odot P_2 \subseteq P_1 \cup P_2$ and $\{q\} \in \mathcal{AS}(P_1 \odot P_2)$ is a subset of $\{p, q\} \in \mathcal{AS}(P_1 \cup P_2)$.

5 Permissible Composition

In Section 3, we introduced the compositional semantics of two programs and Section 4 provided a method of composing programs. In this section, we argue permissible conditions for the compositional semantics in multi-agent coordination. First, we introduce a criterion for selecting answer sets in the compositional semantics.

Definition 5.1. Let P_1 and P_2 be two consistent programs, and Q a result of composition. Then, any answer set $S \in AS(Q)$ is *conservative* if it satisfies every rule in $P_1 \cup P_2$.

Example 5.1. Recall two programs in Example 4.1,

$$\begin{aligned} P_1 :\ & p \leftarrow not\, q, \\ & q \leftarrow not\, p, \\ & s \leftarrow p, \\ P_2 :\ & p \leftarrow not\, r, \\ & r \leftarrow not\, p, \end{aligned}$$

where $\mathcal{AS}(P_1) = \{\{p, s\}, \{q\}\}$ and $\mathcal{AS}(P_2) = \{\{p\}, \{r\}\}$. The compositional semantics is $\mathcal{AS}(Q) = \{\{p, q\}, \{p, s\}, \{q, r\}\}$. Among them, $\{p, s\}$ and $\{q, r\}$ satisfy every rule in $P_1 \cup P_2$, so they are conservative. Note that $\{p, q\}$ does not satisfy the third rule of P_1.

Conservative answer sets are acceptable to each agent because they satisfy the original program of each agent. Unfortunately, conservative answer sets do not always exist in the compositional semantics. For instance, in Example 5.1 if P_2 contains constraints $\leftarrow s$ and $\leftarrow q$, no conservative answer set exists. Existence of no conservative answer set is not a serious flaw in the compositional semantics, however. In fact, different agents have different beliefs in the multi-agent environment, and it may happen that one agent must give up some original belief to reach a reasonable compromise. On the other hand, an agent may possess some *persistent* beliefs that cannot be abandoned. Those persistent beliefs are retained by each agent in coordination. Formally, those beliefs in a program P are distinguished as $PB \subseteq P$ where PB is the set of rules that should be satisfied by the compositional semantics. In this setting, a variant of the compositional semantics is defined as follows.

Definition 5.2. Let P_1 and P_2 be two consistent programs, and PB_1 and PB_2 their persistent beliefs, respectively. A program Ω is called a *permissible composition* of P_1 and P_2 if it satisfies the condition

$$\mathcal{AS}(\Omega) = \{\, S \mid S \in min(\mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2)) \text{ and } S \text{ satisfies } PB_1 \cup PB_2 \}.$$

The set $\mathcal{AS}(\Omega)$ is called the *permissible compositional semantics* of P_1 and P_2. Any answer set in $\mathcal{AS}(\Omega)$ is called a *permissible answer set*. By the definition, permissible composition adds an extra condition to the compositional semantics of Definition 3.1. The permissible compositional semantics reduces to the compositional semantics when $PB_1 \cup PB_2 = \emptyset$. In particular, conservative answer sets are permissible answer sets with $PB_1 \cup PB_2 = P_1 \cup P_2$.

Every permissible answer set satisfies persistent beliefs of each agent, and extends a belief set of an agent by additional information of another agent. Since permissible answer sets are answer sets of the compositional semantics, they inherit properties provided in Section 3 (except Proposition 3.3(2)).

Program composition that reflects the permissible compositional semantics is achieved by introducing every rule in $PB_1 \cup PB_2$ as a constraint to $P_1 \odot P_2$. Given a program P, let $IC(P) = \{ \leftarrow \mathit{body}(r), \mathit{not\_head}(r) \mid r \in P \}$ where $\mathit{not\_head}(r)$ is the conjunction of NAF-literals $\{ \mathit{not}\, L_1, \ldots, \mathit{not}\, L_l \}$ for $\mathit{head}(r) = \{ L_1, \ldots, L_l \}$.

Theorem 5.1. *Let P_1 and P_2 be consistent programs, and Ω a result of permissible composition. Then, $\mathcal{AS}(\Omega) = \mathcal{AS}((P_1 \odot P_2) \cup IC(PB_1) \cup IC(PB_2))$.*

Proof. By the definition of $\mathcal{AS}(\Omega)$ and the result of Theorem 4.4, $S \in \mathcal{AS}(\Omega)$ iff S is an answer set of $P_1 \odot P_2$ and satisfies $PB_1 \cup PB_2$
iff S is an answer set of $P_1 \odot P_2$ and satisfies $IC(PB_1) \cup IC(PB_2)$
iff $S \in \mathcal{AS}((P_1 \odot P_2) \cup IC(PB_1) \cup IC(PB_2))$. $\square$

Example 5.2. Consider two programs in Example 5.1 where $PB_1 = \{ s \leftarrow p \}$ and $PB_2 = \emptyset$. Then, $(P_1 \odot P_2) \cup IC(PB_1) \cup IC(PB_2)$ becomes

$$\begin{aligned} &p \,;\, q \leftarrow, \\ &p \,;\, r \leftarrow, \\ &q \,;\, s \leftarrow p, \\ &\leftarrow p,\, \mathit{not}\, s, \end{aligned}$$

which has two permissible answer sets $\{p, s\}$ and $\{q, r\}$.

6 Discussion

A lot of studies exist for compositional semantics of logic programs (see [6, 9] for excellent surveys). A semantics is *compositional* if the meaning of a program can be obtained from the meaning of its components. The union of programs is the simplest composition between programs. However, semantics of logic programs is not compositional with respect to the union of programs even for definite logic programs. For instance, two definite logic programs $P_1 = \{ p \leftarrow q \}$ and $P_2 = \{ q \leftarrow \}$ have the least Herbrand models $\emptyset$ and $\{q\}$, respectively. But the least Herbrand model of the program union $P_1 \cup P_2$ is not obtained by the composition of $\emptyset$ and $\{q\}$. To solve the problem, a number of different compositional semantics have been proposed in the literature [6]. In composing nonmonotonic logic programs, difficulty of the problem is understood as: "*non-monotonic reasoning and compositionality are intuitively orthogonal issues that do not seem* *easy to be reconciled. Indeed the semantics for extended logic programs are typically non-compositional w.r.t. program union*" [6]. With this reason, studies for compositional semantics of nonmonotonic logic programs mainly concern with the issue of devising a compositional semantics that can accommodate (restricted) nonmonotonicity, or imposing syntactic conditions on programs to be compositional [5, 7, 8, 10, 15].

In this respect, our approach is different from those previous studies. Our primary interest is not simply merging two programs but building a new program that combines answer sets of the original programs. One may wonder the practical value of such combination of answer sets aside from original programs. For instance, given two programs $P_1 = \{ \neg p \leftarrow \mathit{not}\, p \}$ and $P_2 = \{ p \leftarrow \}$, one would consider the meaning of program composition as the answer set $\{p\}$ of $P_1 \cup P_2$. By contrast, our compositional semantics $P_1 \odot P_2$ becomes inconsistent, i.e., combination of $\{\neg p\}$ and $\{p\}$ produces Lit. To justify our position, suppose the following situation: the agent P_1 does not believe the existence of an alien unless its existence is proved, while the agent P_2 believes the existence of aliens with no doubt. The situation is encoded by the above program. Then, what conclusion should be drawn after combining these conflicting beliefs of agents? If one simply merges beliefs by program union, the existence of alien is concluded by the answer set $\{p\}$. In our compositional semantics, two beliefs do not coexist thereby contradict. In multi-agent environments, different agents have different levels of beliefs. A cautious agent might have knowledge in a default form, while an optimistic agent might have knowledge in a definite form. In this circumstance, it appears careless to simply merge knowledge from different information sources. We then took an approach of retaining belief of each agent and combine answer sets of different programs. As a result, the compositional semantics reflects information included in (at least one) answer set of the original programs. In this sense, our program composition is intended to coordinate agents, rather than to synthesize a program by its component. Note that program composition should be distinguished from *revision* or *update*, in which one of two information is known more reliable. In the above example, it is reasonable to accept $P_1 \cup P_2$ as a result of revision/update of P_1 with P_2. Because in this case P_2 is considered new information which precedes P_1. In program composition P_1 and P_2 are supposed to have the same status, so there is no reason to rely P_2 over P_1.

Baral *et al.* [1] introduce algorithms for combining logic programs by enforcing satisfaction of integrity constraints. They request that every answer set of a resulting program to be a subset of an answer set of $P_1 \cup P_2$, which is different from our requirement. Their algorithm is not applicable to unstratified logic programs. The compositional semantics introduced in this paper does not enforce satisfaction of integrity constraints of original programs. One reason for this is that in nonmonotonic logic programs inconsistency may arise aside from integrity constraints. For instance, the integrity constraint $\leftarrow p$ has the same effect as the rule $q \leftarrow p,\ \mathit{not}\, q$ under the answer set semantics. Then, there seems no reason to handle integrity constraints exceptionally in a program. If desired, however, it is easy to have a variant of program composition satisfying constraints as $(P_1 \odot P_2) \cup IC_1 \cup IC_2$, where IC_i $(i = 1, 2)$ is the set of integrity constraints included in P_i. By the introduction of integrity constraints, every answer set which does not satisfy $IC_1 \cup IC_2$ is filtered out. This is also realized by a permissible version of the compositional semantics by putting $PB_1 = IC_1$ and $PB_2 = IC_2$. Combination of propositional theories has also been studied under the names of *merging* [12] or *arbitration* [13], but they do not handle nonmonotonic theories. Sakama and Inoue [14] introduce a framework of coordination between logic programs. They study two problems as follows: given two programs P_1 and P_2, (i) find a program Q which has the set of answer sets such that $\mathcal{AS}(Q) = \mathcal{AS}(P_1) \cup \mathcal{AS}(P_2)$; and (ii) find a program R which has the set of answer sets such that $\mathcal{AS}(R) = \mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)$. A program Q is called *generous coordination* and R is called *rigorous coordination* of two programs. They provide methods of building such programs. Compared with the program composition of this paper, generous/rigorous coordination does not change answer sets of the original programs. That is, generous one collects every answer set of each program, while rigorous one picks up answer sets that are common between two programs. By contrast, we combine answer sets of each program in every possible way. The resulting program and its compositional semantics are both different from generous/rigorous coordination. As addressed above, our program composition is also intended to coordinate agents, it would be interesting to investigate relations among those different types of coordination.

The program composition introduced in Section 4 produces NAF-free EDPs. One may think this uneasy, because this is the case even for composing ELPs containing no disjunction. Disjunctive programs are generally harder to compute, so that it is desirable to have a non-disjunctive program as a result of composing non-disjunctive programs. Technically, the program $P_1 \odot P_2$ is transformed to a non-disjunctive program if $P_1 \odot P_2$ is *head-cycle-free*, i.e., it contains no positive cycle through disjuncts appearing in the head of a disjunctive rule [3]. If $P_1 \odot P_2$ is head-cycle-free, the program is converted to an ELP by shifting disjuncts in the head of a rule to the body as NAF-literals in every possible way as leaving one in the head. For instance, the program $P_1 \odot P_2$ in Example 4.1 is converted to the ELP: $\{\, p \leftarrow not\, q,\;\; q \leftarrow not\, p,\;\; p \leftarrow not\, r,\;\; r \leftarrow not\, p,\;\; q \leftarrow p,\, not\, s,\;\; s \leftarrow p,\, not\, q \,\}$. The resulting program has the same answer sets as the original disjunctive program.

7 Conclusion

This paper has studied compositional semantics of nonmonotonic logic programs. Given two programs, we first introduced combination of answer sets as the compositional semantics of those programs. Then, we developed a method of building a program which reflects the compositional semantics of the original programs. A permissible composition was also introduced for multi-agent coordination. The proposed framework provides a new compositional semantics of nonmonotonic logic programs, and serves as a declarative basis for coordination in multi-agent systems. From the viewpoint of answer set programming, program composition is considered as a program development under a specification that requests a program reflecting the meanings of two or more programs.

The approach taken in this paper requires computing every answer set of programs before composition. This may often be infeasible when a program possesses an exponential number of answer sets. The same problem arises in computing answer sets by existing answer set solvers, and to overcome the bottleneck some approximation techniques would be required. Combining nonmonotonic theories is difficult but important research topic in logic based multi-agent systems, and there is much work to be done.

References

1. C. Baral, S. Kraus, and J. Minker. Combining multiple knowledge bases. *IEEE Transactions of Knowledge and Data Engineering*, 3(2):208–220, 1991.
2. C. Baral and M. Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19/20:73–148, 1994.
3. R. Ben-Eliyahu and R. Dechter. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 12(1):53–87, 1994.
4. S. Brass and J. Dix. Characterizations of the disjunctive stable semantics by partial evaluation. *Journal of Logic Programming*, 32(3):207–228, 1997.
5. A. Brogi, S. Contiero, and F. Turini. Programming by combining general logic programs. *Journal of Logic and Computation*, 9(1):7–24, 1999.
6. A. Brogi. On the semantics of logic program composition. *Program Development in Computational Logic, Lecture Notes in Computer Science*, 3049, pp. 115–151, Springer, 2004.
7. A. Brogi, E. Lamma, P. Mancarella, and P. Mello. A unifying view for logic programming with nonmonotonic reasoning. *Theoretical Computer Science*, 184(1):1–59, 1997.
8. F. Bry. A compositional semantics for logic programs and deductive databases. *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pp. 453–467, MIT Press, 1996.
9. M. Bugliesi, E. Lamma, and P. Mello. Modularity in logic programming. *Journal of Logic Programming*, 19/20:443–502, 1994.
10. S. Etalle and F. Teusink. A compositional semantics for normal open programs. *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pp. 468–482, MIT Press, 1988.
11. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–385, 1991.
12. S. Konieczny and R. Pino-Pérez. On the logic of merging. *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning*, pp. 488–498, Morgan Kaufmann, 1998.
13. P. Liberatore and M. Schaerf. Arbitration (or how to merge knowledge bases). *IEEE Transactions on Knowledge and Data Engineering* 10(1):76–90, 1998.
14. C. Sakama and K. Inoue. Coordination between logical agents. *Proceedings of the 5th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA-V)*, Lecture Notes in Artificial Intelligence, 3487. pp. 161–177, 2005.
15. S. Verbaeten, M. Denecker, and D. De. Schreye. Compositionality of normal open logic programs. *Proceedings of the 1997 International Symposium on Logic Programming*, pp. 371–385, MIT Press, 1997.

Speculative Constraint Processing with Iterative Revision for Disjunctive Answers

Martine Ceberio[1], Hiroshi Hosobe[2], and Ken Satoh[2]

[1] University of Texas at El Paso
500 West University Avenue, El Paso, Texas 79968-0518, USA
mceberio@cs.utep.edu
[2] National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
{ ksatoh, hosobe }@nii.ac.jp

Abstract. In multi-agents systems, incompleteness, due to either communication failure or response delay, is a major problem to handle. To face incompleteness, frameworks for speculative computation were proposed (see [5, 6, 4]). The idea developed in such frameworks is to allow the asking agent, while waiting for slave agents to reply, to reason using default belief until replies are sent.
In [6] in particular, a framework is proposed, that allows an agent not only to perform speculative computation but also to accept iterative answer revision, in the case of yes/no questions. In this paper, we present an extension of the framework in the case of more general types of questions using constraint logic programming (CLP).

1 Introduction

Multi-agent systems are very fashionable and convenient, for they make it possible, for instance, to take advantage of multi-processor machines, and for they also make it possible to design human-like efficient organizations of agents. The main limitation to such an approach is that, as arises in human organizations, communication may be an issue: delayed or broken, it leads to incompleteness of the information in the reasoning structure.

This is a concrete concern when we consider distributed systems such as the Internet, in which communication is indeed not guaranteed, and even if we could guarantee it, communication may either take time, or agents themselves may delay their sending information.

In the case of such unideal, but as we believe, practical situations, when problem-solving is at stake, frameworks for speculative computations were proposed: first for yes/no questions only [5], and then for general questions [4] using constraints.

In [5] and [4], they only provide the possibility for the master agent to perform speculations and a returned answer from the slave agent is final and there is no possibility of change of answers. However, if we let every agent perform speculative computation, the asked agent may revise his answer since the previous answer sometimes depends on the asked agent's belief, which might turn out to be false. Therefore, a chain reaction of belief revision among agents might occur which was firstly observed in [6], and Satoh and Yamamoto provide a revisable speculative computation method for yes/no questions. Essential part of their work is a dynamic iterative belief revision mechanism which can handle a revision of an answer for query even during the execution.

Belief revision is indeed very important for both the sake of flexibility (information is processed before it is complete), and speed of computation (time is saved in case prior information is later entailed).

In this paper, we combine the methods proposed in [4] and [6], and extend them, so that we can handle iterative answer revision for a query with constraints. We also complete these methods with the ability to corporate disjunctive answers. So, the main contribution of this paper is the definition of a framework that enables to perform speculative computations on constraints while handling belief revision, and that handles as well disjunctive answers. In particular, the main challenges dealt with in this work are the following.

– First, processing speculative constraints, as shown in [4], is manageable when belief revision is not considered. In this paper, belief revision is made possible because it enables more speculative computation in multi-agent systems. This hardens the problem a lot: the process management needs to be modified so as to enable changes in the computation at any time, while maintaining a reasonable balance between not being too much space-consuming, and not loosing too much time (*i.e.*, we don't want to start from scratch all the time). The process management is presented in detail in this paper, as well as results on the space complexity of our operational model.

– The second challenging point described in this paper is the way disjunction is now handled in the framework we propose. Indeed, considering the situation where each agent's behavior is specified as a CLP program, we need to handle alternative answers, since these answers may come from different derivations in CLP. By manipulating such alternative answers, we face another complication, in that we need to distinguish a revised answer of a previous answer, from an answer derived from an alternative derivation path[3]. To solve this problem, we devise an answer entry which keeps track of the usage status of the answer in processes. This new feature impacts the way processes are managed, as described in Section 3, and therefore makes the problem more complicated.

For an iterative belief revision, many proposals have been described. As far as we know, existing frameworks separate reasoning and belief revision, except [5, 6, 4]. And this work is along the line of the works of Satoh et al. in a more general setting.

There are works on a formalization of an agent in terms of logic programming such as [3]. Although these research are important in their own right, our paper

[3] Indeed, in particular, a contradictory answer should be considered as contradictory only if it is a revision of a former answer, not if it is an alternative answer.

pursues another branch of investigation in the context of speculative computation.

Most related research would be constraint programming language such as AKL(Andorra Kernel Language) [2] and Oz [7] which perform a kind of speculative computation. AKL allows local speculative variable bindings in a guard of each clauses until one of guards is succeeded and Oz can control multiple computation spaces each of which represents alternative path of constraint processing. As far as we understand, however, speculative computation used in these languages are mainly motivated for or-parallel computing where all alternative paths of computation are executed in parallel until one of paths are succeeded eventually. On the other hand, we regard a speculative computation as a default computation where most plausible paths of computation are executed. Moreover, they do not consider any revision of the answers.

The structure of the paper is as follows. We firstly define a framework for speculative constraint processing and a semantics of the framework. Then, we describe an operational model and show an example of execution and state correctness of our model. Finally, we discuss space complexity issues, before to conclude.

2 Speculative Constraint Processing

In this section, we provide a framework of speculative constraint computation based on the CLP framework [1]. This framework is designed so that an agent not only performs speculative constraint processing but also accepts revised answers and alternative answers. We then define a semantics of this framework, in Subsection 2.2.

2.1 Framework Definition

Definition 1. *Let Σ be a finite set of constants. We call an element in Σ a* slave agent identifier*. An* atom *is of the form either $p(t_1, ..., t_n)$ or $p(t_1, ..., t_n)@S$ where p is a predicate, $t_i (1 \leq i \leq n)$ is a term, and S is in Σ.*

We call an atom with an agent identifier an "*askable atom*", and an atom without an identifier a "*non-askable atom*".

Definition 2. *A* framework for speculative constraint computation, in a master-slave system *is a triple $\langle \Sigma, \Delta, \mathcal{P} \rangle$ where:*

- *Σ is a finite set of constants;*
- *Δ is a set of rules of the following form called* default rule *w.r.t. $Q@S$:*

$$Q@S \leftarrow C\|.$$

where $Q@S$ is an askable atom, each of whose arguments is a variable, and C is a set of constraints, called default constraint for $Q@S$;

– $\mathcal{P}$ *is a constraint logic program, that is, a set of rules of the form:*

$$H \leftarrow C \| B_1, B_2, ..., B_n.$$

where:

- *H is a non-askable atom; we refer to H as the* head *of R denoted as $head(R)$;*
- *C is a set of constraints, called the* constraint *of R, and denoted as $const(R)$;*
- *each B_i of $B_1, ..., B_n$ is either an askable atom or a non-askable atom, and we refer to $B_1, ..., B_n$ as the* body *of R denoted as $body(R)$.*

Note that a default is not necessarily specified for every askable atom. Moreover, we allow multiple defaults for the same askable atom.

Example 1. We consider the following example of hotel room reservation. There is a master agent m: m asks travellers a and b. If both travel, m reserves a twin room. If one of them travels, m reserves a single room. Agent m has default information about the status of a and b for days 1, 2 and 3, but the real status will be obtained directly from a and b, and the status is therefore likely to be changed.

This example can be represented as the following multi-agent system $\langle \Sigma, \Delta, \mathcal{P} \rangle$[4]:

– Σ is the set of slave agents. Here, there is one master agent, m, and two slave agents, a and b. Therefore $\Sigma = \{a, b\}$.

– Δ is the set of default information (default rules), assumed by the master agent. In particular, let us suppose that m assume that a is free on days 1, and 2, and busy on day 3, and that b is free on day 2, and busy on day 1. Then the corresponding set Δ is as follows:

$$\begin{aligned} \Delta = \{ \; & d_1: \; fr(D)@a \leftarrow D=1 \|., \\ & d_2: \; fr(D)@a \leftarrow D=2 \|., \; d_3: \; bs(D)@a \leftarrow D=3 \|., \\ & d_4: \; fr(D)@b \leftarrow D=2 \|., \; d_5: \; bs(D)@b \leftarrow D=1 \|. \} \end{aligned}$$

Let us remark that it is not necessary that a default information exist for all cases. In particular, m has no default information concerning the status of b on day 3.

– $\mathcal{P}$ is a constraint logic program, to be solved by agent m. In our case of hotel room reservation with two travelers, it is made of the following set of rules:

$$\begin{aligned} rsv(R, L, D) &\leftarrow R=tr, L=[a,b] \| fr(D)@a, fr(D)@b. \\ rsv(R, L, D) &\leftarrow R=sr, L=[a] \| fr(D)@a, bs(D)@b. \\ rsv(R, L, D) &\leftarrow R=sr, L=[b] \| bs(D)@a, fr(D)@b. \end{aligned}$$

In order to solve this constraint satisfaction problem, agent m will have to ask agents a and b about $fr(D)@a$, $bs(D)@a$, $fr(D)@b$, $bs(D)@b$.

[4] A string beginning with an upper case letter represents a variable and a string beginning with a lower case letter represents a constant. We abbreviate "free" as fr, "busy" as bs, "travel" as $trvl$, "reserve" as rsv, "twin room" as tr, and "single room" as sr.

2.2 Semantics of Speculative Constraint Processing

For a semantics of the above framework, we index the semantics of constraint logic program by a *reply set* which specifies a reply for an askable atom.

Definition 3. *A* reply set *is a set of rules of the form:*

$$Q@S \leftarrow C\|,$$

where $Q@S$ is an askable atom, each of whose arguments is a variable, and C is a constraint over these variables.

Let $\langle \Sigma, \Delta, \mathcal{P} \rangle$ be a framework for speculative constraint computation, and $\mathcal{R}$ be a reply set. A belief state *w.r.t. $\mathcal{R}$ and Δ is a reply set defined as:*

$$\mathcal{R} \cup \{ \text{“}Q@S \leftarrow C\|\text{”} \in \Delta \mid \neg\exists\ C' \text{ s.t. “}Q@S \leftarrow C'\|\text{”} \in \mathcal{R}\}$$

and denoted as $BEL(\mathcal{R}, \Delta)$.

We introduce the above belief state, since if the answer is not returned, we use a default rule for an unreplied askable atom.

Definition 4. *A goal is of the form $\leftarrow C\|B_1, ..., B_n$ where C is a set of constraints and B_i's are atoms. We call C the* constraint of the goal *and $B_1, ..., B_n$ the* body of the goal.

Definition 5. *A reduction of a goal $\leftarrow C\|B_1, ..., B_n$ w.r.t. a constraint logic program $\mathcal{P}$, a reply set $\mathcal{R}$ and an atom B_i, is a goal $\leftarrow C'\|B'$ such that:*

- *there is a rule R in $\mathcal{P} \cup \mathcal{R}$ s.t. $C \wedge (B_i = head(R)) \wedge const(R)$ is consistent*[5].
- *$C' = C \wedge (B_i = head(R)) \wedge const(R)$*
- *$B' = \{B_1, ...B_{i-1}, B_{i+1}, ..., B_n\} \cup body(R)$*

Definition 6. *A derivation of a goal $G = \leftarrow C\|Bs$ w.r.t. a framework for speculative constraint computation $\mathcal{F} = \langle \Sigma, \Delta, \mathcal{P} \rangle$ and a reply set $\mathcal{R}$ is a sequence of reductions “$\leftarrow C\|Bs$”,..., “$\leftarrow C'\|\emptyset$”*[6] *w.r.t. $\mathcal{P}$ and $BEL(\mathcal{R}, \Delta)$ where in each reduction step, an atom in the body of the goal in each step is selected. C' is called an* answer constraint *w.r.t. G, $\mathcal{F}$ and $\mathcal{R}$. We call a set of all answer constraints w.r.t. G, $\mathcal{F}$ and $\mathcal{R}$ the* semantics of G *w.r.t. $\mathcal{F}$ and $\mathcal{R}$.*

In the above definition, we only consider the most recent reply set, whereas a reply set might be varied during execution according to the slave agent's answer revision. We use the most recent reply set because it reflects the current situation of the slave agents.

[5] A notation $B_i = head(R)$ represents a conjunction of constraints equating the arguments of atoms B_i and $head(R)$.
[6] $\emptyset$ denotes an empty goal.

3 An Operational Model for Speculative Computation with Iterative Answer Revision

3.1 Overview of Operational Model

The execution of the speculative framework is based on two phases, a *process reduction phase* and a *fact arrival phase.* The process reduction phase is a normal execution of a program in a master agent, and the fact arrival phase is an interruption phase when an answer arrives from a slave agent.

For the operational model, we use the following two kinds of objects: a *process* and an *answer entry.*

Each *process* represents an alternative way of computation. Processes are created when a choice point of computation is encountered, such as case splitting, default handling and answer arrival. A process becomes a finished process when the body of the associated goal with the process becomes empty. A process fails when some used default constraints are found to contradict the newly returned answer.

An *answer entry* is used to distinguish alternative answers and to detect which old answer corresponds to the newly revised answer. This detection is done by attaching an ID to each answer. If a new answer with an ID different from any existing answer comes, it is an alternative answer. Otherwise, the new answer is considered as a revised answer for the old answer with the same ID.

Figures 1∼4 intuitively explain how processes are updated according to askable atoms. In the tree, each node represents a process, but we only show constraints associated with the process. The top node represents a constraint for the original process, and the other nodes represent added constraints for the reduced processes. The leaves of the process tree represent the current processes. Therefore, the processes which are not in the leaves are deleted processes.

Fig. 1 shows a situation of the processes represented as a tree when an askable atom, whose reply has not arrived yet, is executed in the process reduction phase. In this case, the current process, represented by the processed constraints C, is splitted into two different kinds of processes: the first one is a process using default information, C_d, and is called *default process* [7]; and the other one is the current process C itself, called *original process*, suspended at this point.

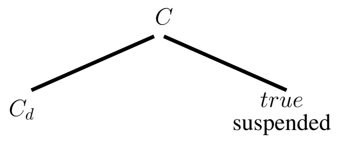


Fig. 1. When $Q@S$ is processed, during the process reduction phase

Note that, if there are multiple definitions of defaults, we will have more than one default process, but still only one suspended process. In addition, let us note

[7] In this figure, we assume that there is only one default for brevity.

that the reason for suspending processes (which is, keeping them in memory), is that in case of a contradictory revision of the default, or later alternative answers coming, it is essential to keep memory of the original processes to be able to restore them.

When, after some reduction of the default processes (represented on Fig. 2 by dashed lines), the first answer comes from a slave agent, expressing constraint C_f for this askable literal, we update default processes as well as the original suspended process as follows:

- Default process(es) are reduced into two different kinds of processes: the first kind is a process adding C_f to the problem to solve, and the other is the current process itself which is suspended at this point[8].
- The original process is reduced into two different kinds of processes as well: the first kind is a process adding $\neg C_d \wedge C_f$, and the other is the original process, suspended at this point.

Let us remark that although the tree of processes grows, only leaves are kept in memory.

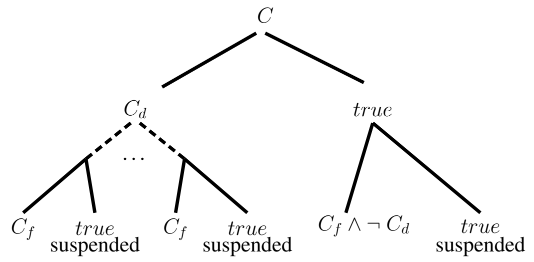


Fig. 2. When the first answer C_f for $Q@S$ arrives

To explain the correctness of the above process update intuitively, we define a *frontier* which represents the computation status of all alternative derivations. A *frontier* w.r.t. a goal $\leftarrow C \| Bs$, a framework for speculative constraint computation $\langle \Sigma, \Delta, \mathcal{P} \rangle$ and a reply set $\mathcal{R}$, is a set of goals defined as follows.

1. The set consisting of the initial goal, $\{\leftarrow C \| Bs\}$ is a frontier.
2. Let F be a frontier w.r.t. the above initial goal, the framework and the reply set. If a goal G is in F, B is an atom in G, and $RGs = \{G' \mid G'$ is a reduction of G w.r.t. $\mathcal{P}$, $BEL(\mathcal{R}, \Delta)$ and $B\}$, then $F \backslash \{G\} \cup RGs$ is a frontier.

Then we have the following properties.

Lemma 1. *Let $\leftarrow C \| Bs$ be a goal, F be a frontier of this goal, and C' be a constraint. If we add C' to the constraints of every goal in F, then the disjunctions of all answer constraints of these modified goals is logically equivalent to the disjunction of all answer constraints of the goal $\leftarrow C \wedge C' \| Bs$.*

[8] Let us remark that this splitting process is similar to the splitting process above-described for the case of a first default used.

Lemma 2. *Let $\leftarrow C \| Bs$ be a goal, $\mathcal{R}$ be a reply set, and C' be a constraint. Then, the disjunction of answer constraints of $\leftarrow C \wedge C' \| Bs$ and $\leftarrow C \wedge \neg C' \| Bs$ is logically equivalent to the disjunction of all answer constraints of $\leftarrow C \| Bs$.*

Let $\leftarrow C \| Bs$ be a goal containing $Q@S$, suppose that it is reduced into $\leftarrow C \wedge C_d \| Bs \backslash \{Q@S\}$ by a default rule "$Q@S \leftarrow C_d \|$". Let F be a frontier of $\leftarrow C \wedge C_d \| Bs \backslash \{Q@S\}$ when the first reply "$Q@S \leftarrow C_f \|$" is returned. Since our semantics considers the most recent replies, at this point, we should consider:

$$\leftarrow C \wedge C_f \| Bs \backslash \{Q@S\}$$

instead of:

$$\leftarrow C \wedge C_d \| Bs \backslash \{Q@S\}.$$

One possibility to implement this change is that we just discard F and invoke a new goal $\leftarrow C \wedge C_f \| Bs \backslash \{Q@S\}$. However, in this case, we throw every computation away before F is obtained. To retain the previous computation as much as possible, we propose the following execution.

1. We add C_f to the constraint of every goal in F.
   Let us remark that the disjunction of all answer constraints from this new frontier is logically equivalent to the disjunction of all answer constraints of $\leftarrow C \wedge C_d \wedge C_f \| Bs \backslash \{Q@S\}$ as Lemma 1 states. This computation keeps the previous computation which is consistent with the new reply (C_f).

2. In addition to the above computation, we also start computing a new goal:

   $$\leftarrow C \wedge \neg C_d \wedge C_f \| Bs \backslash \{Q@S\}$$

   to guarantee completeness. It is because the disjunction of all answer constraints derived from $\leftarrow C \wedge C_d \wedge C_f \| Bs \backslash \{Q@S\}$ and $\leftarrow C \wedge \neg C_d \wedge C_f \| Bs \backslash \{Q@S\}$ is logically equivalent to the disjunction of all answer constraints derived from $\leftarrow C \wedge C_f \| Bs \backslash \{Q@S\}$ as Lemma 2 states.

When an alternative answer, with the constraint C_a, comes from a slave agent (as shown on Fig. 3), we need to follow the same procedure as when the first answer comes (*cf.* Fig. 2), except that now the processes handling only default information are suspended. So, this is done by splitting the suspended default process(es), in order to obtain the answer constraints which are logically equivalent to the answer constraints of:

$$\leftarrow C \wedge C_d \wedge C_a \| Bs \backslash \{Q@S\},$$

as well as by splitting the suspended original process, in order to obtain the answer constraints which are logically equivalent to the answer constraints of $\leftarrow C \wedge \neg C_d \wedge C_a \| Bs \backslash \{Q@S\}$ (Fig. 3). By gathering these answer constraints, we can compute all answer constraints for the alternative reply.

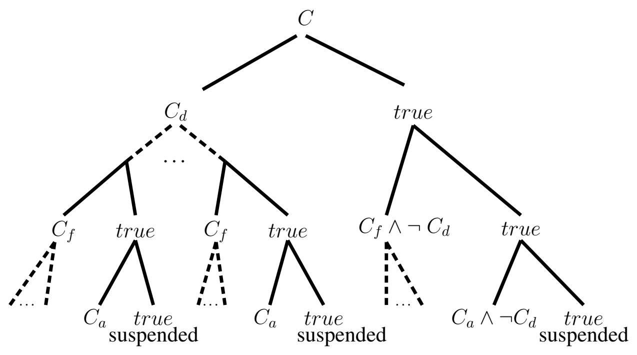


Fig. 3. When the alternative answer C_a for $Q@S$ arrives

On the other hand, when a revised answer, with the constraint C_r, comes, all processes using the first (or current) answer are splitted, in order to obtain the answer constraints which are logically equivalent to the answer constraints of:

$$\leftarrow C \land C_f \land C_r \| Bs \backslash \{Q@S\},$$

and the suspended original process is splitted as well, in order to obtain the answer constraints which are logically equivalent to the answer constraints of $\leftarrow C \land \neg C_f \land C_r \| Bs \backslash \{Q@S\}$ (Fig. 4). By gathering these answer constraints, we can override the previous reply by the revised reply.

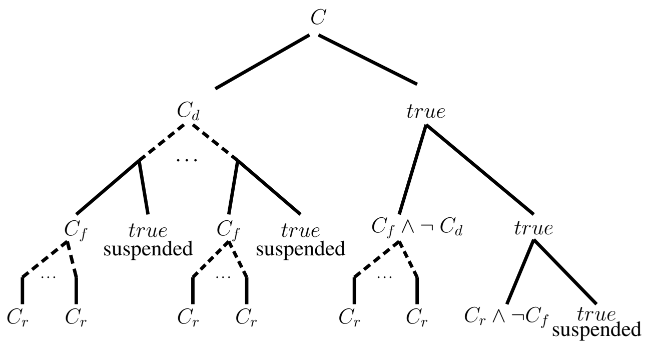


Fig. 4. When the revised answer C_r for $Q@S$ arrives

3.2 Preliminary Definitions

A process is either an *ordinary process* or a *finished process*. An *ordinary process* P is an expression of the form $\langle PID, C, GS, WA, AA \rangle$ where:

- PID: the ID for a process denoted as $pid(P)$;

- C: the current constraint in the goal denoted as $pconst(P)$;
- GS: the body in the goal denoted as $gs(P)$;
- WA: a set of pairs $\langle Q@S, WAID\rangle$ where $Q@S$ is an askable atom and $WAID$ is the ID of an answer entry whose answer is waited for by the process. We denote WA as $wa(P)$.
- AA: a set of pairs $\langle Q@S, AAID\rangle$ where $Q@S$ is an askable atom and $AAID$ is the ID of an answer entry whose answer is used in the process. We denote AA as $aa(P)$.

A *finished process* FP is an expression of the form $\langle Query, FPID, C\rangle$ where:

- $Query$: an initial query for this process. It is used to send an answer to the asking agent;
- $FPID$: the ID for a process. This is also used when this answer is returned to the asking agent;
- C: the current constraint in the process.

For simplicity, an ordinary process is sometimes just called a process.

An *answer entry* A is an expression of the form $\langle Q@S, AID, C, UPIDs\rangle$ where:

- $Q@S$: the query given to the other agent denoted as $aq(A)$;
- AID: the ID for an answer entry denoted as $aid(A)$. We have the special IDs, "o" for the answer entry created when this query is firstly asked, and "$d_1, ...$" for default answers. We call an answer entry with the ID "o" an *original answer entry* for $Q@S$, an answer entry with an ID of "$d_1, ...$" a *default answer entry*, and other answer entries *ordinary answer entries*;
- C: the most recent answer constraint for $Q@S$ for answer entry A denoted as $aconst(A)$. The constraint of the original answer entry is defined as $true$;
- $UPIDs$: the set of IDs of processes using an answer in A denoted as $ups(A)$.

3.3 Process Reduction Phase

In the process reduction phase, we process the constraints we have, in a regular CP way. The only difference is that we may have to consider default information, or answers. In this subsection, we describe how we manage processes, following the above-given definitions.
We do the following until no more process can be processed.

- When a query $Q_{init}@S_{self}$ is asked from another agent S' where S_{self} is the ID for this agent, we record Q_{init} as the initial query and S' as the asking agent. We then create a new process $\langle PID, \{\}, Q_{init}, \{\}, \{\}\rangle$ where PID is a new process ID.
- If there is an ordinary process P such that $gs(P) = wa(P) = \emptyset$,
  1. Send an answer to the asking agent S' which is of the form: $\langle Q_{init}@S_{self}, pid(P), pconst(P)\rangle$.
  2. We change this process into a finished process of the form: $\langle Q_{init}@S_{self}, pid(P), pconst(P)\rangle$.

– Else if there is a process P such that $gs(P) \neq \emptyset$ and $wa(P) = \emptyset$, then we select an atom L in $gs(P)$ and reduce L as follows.
  - If L is a non-askable atom,
    1. For every rule R such that $pconst(P) \wedge (L = head(R)) \wedge const(R)$ is consistent, we do the following:
       (a) We create the following process $\langle newPID, newC, GS, \{\}, AA\rangle$ where
         * $newPID$ is a new process ID;
         * $newC := pconst(P) \wedge (L = head(R)) \wedge const(R)$;
         * $GS := body(R) \cup gs(P) \backslash \{L\}$;
         * $AA := aa(P)$.

       (b) For every answer entry A s.t. $\langle aq(A), aid(A)\rangle$ in $aa(P)$, $ups(A) := ups(A) \cup \{newPID\}$.
    2. For every answer entry A s.t. $\langle aq(A), aid(A)\rangle$ in $aa(P)$, $ups(A) := ups(A) \backslash \{pid(P)\}$.
    3. We delete P.
  - If L is an askable atom $Q@S$,
    1. We do either of the following according to non-arrival/arrival of the answer.
       * If there is no ordinary answer entry of the form $\langle Q@S, AID, C, UPIDs\rangle$, then for each default "$Q@S \leftarrow C_d \|$." such that $pconst(P) \wedge C_d$ is consistent, we do the following:

       (a) We create a new process $\langle newPID, newC, GS, \{\}, AA\rangle$ where
         · $newPID$ is a new process ID.
         · $newC := pconst(P) \wedge C_d$
         · $GS := gs(P) \backslash \{Q@S\}$
         · $AA := aa(P) \cup \{\langle Q@S, d\rangle\}$ where d is an ID for this default.

       (b) We associate the newly created process with a default d of $Q@S$ as follows.
         · If there is a default answer entry $A_d = \langle Q@S, d, C_d, UPIDs_d\rangle$, then $ups(A_d) := UPIDs_d \cup \{newPID\}$.
         · Else if there is no default answer of the form $\langle Q@S, d, C_d, UPIDs_d\rangle$, we create the answer entry $\langle Q@S, d, C_d, \{newPID\}\rangle$.

       (c) For every answer entry A s.t. $\langle aq(A), aid(A)\rangle$ in $aa(P)$, $ups(A) := ups(A) \cup \{newPID\}$.
       * Else if there exists an ordinary answer entry of the form $\langle Q@S, AID, C, UPIDs\rangle$, then for each ordinary answer entry $\langle Q@S, AID, C_a, UPIDs\rangle$ s.t. $pconst(P) \wedge C_a$ is consistent, we do the following:

       (a) We create a new process $\langle newPID, newC, GS, \{\}, AA\rangle$ where
         · $newPID$ is a new process ID.
         · $newC := pconst(P) \wedge C_a$
         · $GS := GS \backslash \{Q@S\}$

        · $AA := aa(P) \cup \{\langle Q@S, AID \rangle\}$.
   (b) For every answer entry A s.t. $\langle aq(A), aid(A) \rangle$ in $aa(P)$, $ups(A) := ups(A) \cup \{pid(P)\}$.
2. We associate P with $Q@S$ as follows.
   * If there is an original answer entry $A_o = \langle Q@S, o, true, UPIDs_o \rangle$, then $ups(A_o) := UPIDs_o \cup \{pid(P)\}$.
   * Else if there is no original answer entry of the form $\langle Q@S, o, true, UPIDs \rangle$, we create an answer entry $\langle Q@S, o, true, \{pid(P)\} \rangle$, and send a question Q to S.
3. $wa(P) := \{\langle Q@S, o \rangle\}$

3.4 Fact Arrival Phase

Suppose that an answer is returned from an agent S for a question $Q@S$ of the form $\langle Q@S, AID, C \rangle$. Then, we do the following after one step of process reduction is finished.

– If there is no answer entry of the form $\langle Q@S, AID, C_f, UPIDs' \rangle$[9],
  1. We create an answer entry $\langle Q@S, AID, C, UPIDs \rangle$ where $UPIDs$ is set to $\emptyset$ initially, but will be incremented as shown below.
  2. For every default answer entry for a default d of the form $\langle Q@S, d, C_d, UPIDs_d \rangle$ and for every process P_d such that $pid(P_d) \in UPIDs_d$, we do the following:
     - If P_d is a finished process of the form $\langle Q_{init}@S_{self}, PID, C_{Final} \rangle$ s.t. $C \wedge C_{Final} \neq C_{Final}$, we send an answer of the form $\langle Q_{init}@S_{self}, PID, C \wedge C_{Final} \rangle$ to the asking agent S'.
     - If P_d is an ordinary process,
       (a) $wa(P_d) := wa(P_d) \cup \{\langle Q@S, d \rangle\}$
       (b) $aa(P_d) := aa(P_d) \backslash \{\langle Q@S, d \rangle\}$
       (c) If $C \wedge pconst(P_d)$ is consistent, we do the following.
          i. We create the following process $\langle newPID, newC, GS, WA, AA \rangle$ where
             * $newPID$ is a new process ID.
             * $newC := C \wedge pconst(P_d)$
             * $GS := gs(P_d)$.
             * $WA = wa(P_d)$
             * $AA = aa(P_d) \cup \{\langle Q@S, AID \rangle\} \backslash \{\langle Q@S, d \rangle\}$
          ii. $UPIDs := UPIDs \cup \{newPID\}$.
  3. Pick up the original answer entry of the form $\langle Q@S, o, true, UPIDs_o \rangle$.
  4. For every process P_o such that $pid(P_o) \in UPIDs_o$ and $C \wedge pconst(P_o) \wedge \bigwedge_{(Q@S \leftarrow C_d \|) \in \Delta} \neg C_d$ is consistent, do the following:
     (a) We create the following process $\langle newPID, newC, GS, WA, AA \rangle$ where
         - $newPID$ is a new process ID.

[9] This means that the arriving answer is an alternative answer to the query $Q@S$.

  - $newC := C \wedge pconst(P_o) \wedge \bigwedge_{(Q@S \leftarrow C_d \|) \in \Delta} \neg C_d$
  - $GS := gs(P_o)$.
  - $WA := wa(P_o) \backslash \{\langle Q@S, o\rangle\}$
  - $AA := aa(P_o) \cup \{\langle Q@S, AID\rangle\}$

  (b) $UPIDs := UPIDs \cup \{newPID\}$.

– Else if there is an answer entry of the form $\langle Q@S, AID, C_f, UPIDs'\rangle$[10],

1. We change $\langle Q@S, AID, C_f, UPIDs'\rangle$ into $\langle Q@S, AID, C, UPIDs\rangle$ where $UPIDs := UPIDs'$ initially but will be incremented/decremented as shown below.
2. For every process P such that $pid(P) \in UPIDs'$ do the following:
   - If P is a finished process of the form $\langle Q_{init}@S_{self}, PID, C_{Final}\rangle$ s.t. $C \wedge C_{Final} \neq C_{Final}$, we send an answer of the form $\langle Q_{init}@S_{self}, PID, C \wedge C_{Final}\rangle$ to the asking agent S'.
   - If P is an ordinary process,
     * If $C \wedge pconst(P)$ is consistent, $pconst(P) := C \wedge pconst(P)$.
     * Otherwise, delete P and $UPIDs := UPIDs \backslash \{pid(P)\}$.
3. Pick up the original answer entry of the form $\langle Q@S, o, true, UPIDs_o\rangle$.
4. For every process P_o such that $pid(P_o) \in UPIDs_o$ and $C \wedge pconst(P_o) \wedge \neg C_f$ is consistent, we do the following:

   (a) We create the following process $\langle newPID, newC, GS, WA, AA\rangle$ where
   - $newPID$ is a new process ID.
   - $newC := C \wedge pconst(P_o) \wedge \neg C_f$
   - $GS := gs(P_o)$.
   - $WA := wa(P_o) \backslash \{\langle Q@S, o\rangle\}$
   - $AA := aa(P_o) \cup \{\langle Q@S, AID\rangle\}$

   (b) $UPIDs := UPIDs \cup \{newPID\}$.

3.5 Execution Trace Example

We show a part of an execution trace for a question $rsv(R, L, D)$ in Example 1. In this trace, we consider a scenario which highlights process updates upon arrivals of an alternative answer and revised answer. We firstly give the initial process $\langle p_0, \{\}, \{rsv(R, L, D)\}, \{\}, \{\}\rangle$.

1. Select process p_0 and reduce it to p_1, p_2, p_3.
   Processes:
   $\langle p_1, \{R=tr, L=[a,b]\}, \{fr(D)@a, fr(D)@b\}, \{\}, \{\}\rangle$,
   $\langle p_2, \{R=sr, L=[a]\}, \{fr(D)@a, bs(D)@b\}, \{\}, \{\}\rangle$,
   $\langle p_3, \{R=sr, L=[b]\}, \{bs(D)@a, fr(D)@b\}, \{\}, \{\}\rangle$

[10] This means that the arriving answer is a revised answer of one of the previous answer to the query $Q@S$.

2. Select p_1, and ask a question $fr(D)@a$, and create answer entries for $fr(D)@a$ and new processes p_4, p_5 for default answers.
Answer entries:
$\langle fr(D)@a, o, true, \{p_1\}\rangle$,
$\langle fr(D)@a, d_1, \{D = 1\}, \{p_4\}\rangle$,
$\langle fr(D)@a, d_2, \{D = 2\}, \{p_5\}\rangle$
Processes: p_2, p_3,
$\langle p_4, \theta_{tr} \cup \{D = 1\}, \{fr(D)@b\}, \{\}, \{\langle fr(D)@a, d_1\rangle\}\rangle$[11],
$\langle p_5, \theta_{tr} \cup \{D = 2\}, \{fr(D)@b\}, \{\}, \{\langle fr(D)@a, d_2\rangle\}\rangle$,
$\langle p_1, \theta_{tr}, \{fr(D)@b\}, \{\langle fr(D)@a, o\rangle\}, \{\}\rangle$

3. Suppose that $\langle fr(d)@a, a_1, \{D = 2\}\rangle$ is returned from the agent a. We suspend p_4 and p_5 since they use a default answer and then create new processes p_6 from p_5 since the default answer used in p_5 is consistent with the returned answer. Note that we create no new process from p_1 since the returned answer contradicts one of negations of default answers.
Answer entries: fra_o, fra_{d_1}, fra_{d_2}[12],
$\langle fr(D)@a, a_1, \{D = 2\}, \{p_6\}\rangle$
Processes: p_1, p_2, p_3,
$\langle p_6, \theta_{tr2}, \{fr(D)@b\}, \{\}, \{\langle fr(D)@a, a_1\rangle\}\rangle$,
$\langle p_4, \theta_{tr1}, \{fr(D)@b\}, \{\langle fr(D)@a, d_1\rangle\}, \{\}\rangle$,
$\langle p_5, \theta_{tr2}, \{fr(D)@b\}, \{\langle fr(D)@a, d_2\rangle\}, \{\}\rangle$[13]

4. Suppose that $\langle fr(D)@a, a_2, \{D = 3\}\rangle$ is returned from the agent a. Since this has the different answer ID from the previous answer in the last step, this answer is an alternative answer. Then, we create a new process from p_1 which is the original process for query $fr(D)@a$. Note that we create no new process from the processes created by default answers for $fr(D)@a$ since this answer contradicts the defaults.
Answer entries: fra_o, fra_{d_1}, fra_{d_2}, fra_{a_1}[14],
$\langle fr(D)@a, a_2, \{D = 3\}, \{p_7\}\rangle$
Processes: $p_1, p_2, p_3, p_4, p_5, p_6$,
$\langle p_7, \theta_{tr} \cup \{D = 3, D \neq 1, D \neq 2\}, \{fr(D)@b\}, \{\}, \{\langle fr(D)@a, a_2\rangle\}\rangle$

5. Suppose that $\langle fr(D)@a, a_1, \{D = 1\}\rangle$ is returned from the agent a. The ID a_1 for the returned answer indicates that this answer is a revised answer for "$D = 2$". Therefore, we revise every process using a_1 which is recorded in the answer entry fra_{a_1}. This is p_6, but its associated constraint is contradictory with the returned answer, and therefore we kill this process. Then, we create a new process p_8 from p_1.

[11] $\theta_{tr} = \{R = tr, L = [a, b]\}$.
[12] $fra_o = \langle fr(D)@a, o, true, \{p_1\}\rangle$,
$fra_{d_1} = \langle fr(D)@a, d_1, \{D = 1\}, \{p_4\}\rangle$,
$fra_{d_2} = \langle fr(D)@a, d_2, \{D = 2\}, \{p_5\}\rangle$.
[13] $\theta_{tr2} = \theta_{tr} \cup \{D = 2\}$ and $\theta_{tr1} = \theta_{tr} \cup \{D = 1\}$.
[14] $fra_{a_1} = \langle fr(D)@a, a_1, \{D = 2\}, \{p_6\}\rangle$.

Answer entries: fra_o, fra_{d_1}, fra_{d_2}, fra_{a_2}[15],
$\langle fr(D)@a, a_1, \{D = 1\}, \{p_8\}\rangle$
Processes: $p_1, p_2, p_3, p_4, p_5, p_7$,
$\langle p_8, \theta_{tr} \cup \{D = 1, D \neq 2\}, \{fr(D)@b\}, \{\}, \{\langle fr(D)@a, a_1\rangle\}\rangle$

4 Correctness of the Operational Model

We guarantee that the above operational model gives a correct answer w.r.t. the most recent replies. Let us note that we assume that the order of reply messages is preserved.

Theorem 1. *Let $\langle \Sigma, \Delta, \mathcal{P}\rangle$ be a framework for speculative constraint computation. Suppose that there is an ordinary process P such that $gs(P) = wa(P) = \emptyset$ for the initial query Q_{init}. Let*

$$\mathcal{R} = \{\text{“}Q@S \leftarrow C\|\text{”} \mid \textit{there exists an answer entry } \langle Q@S, AID, C, UPIDs\rangle \\ \textit{s.t. } \langle Q@S, AID\rangle \in aa(P)\}.$$

Then, there exists an answer constraint C' w.r.t. Q_{init}, the framework and $\mathcal{R}$ s.t. $\pi_V(pconst(P))$ entails $\pi_V(C')$, where V is the set of the variables that occur in Q_{init}, and π_V is the projection of constraints onto V.

5 Space complexity of our approach

Our approach, compared to traditional approaches (no belief revision), generates an additional cost in terms of space. In this section, we briefly show that the additional cost in space is linear. This cost is observed based on the size of the set PS of processes related to the revised or alternative answer to handle.

When a revised answer comes, say C_r, as shown in Fig. 4:

- if C_r entails the previous answer, say C_f, PS either remains the same size, or reduces (because some processes in PS may now have inconsistent constraints and therefore be killed);
- if C_r is inconsistent with C_f, then all the processes using C_f in PS are killed, the original suspended processes are duplicated and resumed with C_r, and therefore PS grows by at most the number of original suspended processes;
- if C_r is consistent with C_f but does not entail it, PS grows by at most the number of original suspended processes.

These three cases exhibit only linear (or less) behavior.

When an alternative answer comes, say C_a, as shown in Fig. 3, all the suspended processes created on the arrival of the first answer, as well as the original suspended processes, are duplicated and resumed with C_a. Therefore, PS grows by at most the number of these suspended processes.

As briefly covered here, the growth of the set of processes on the arrival of revised and alternative answers follows a linear behavior.

[15] $fra_{a_2} = \langle fr(D)@a, a_2, \{D = 3\}, \{p_7\}\rangle$.

6 Conclusion

In this paper, we presented an operational model for speculative constraint processing with iterative revision for alternative answers. This paper is a generalization of two previous works; the work of revisable speculative computation for yes/no questions [6] and the work of non-revisable speculative computation for queries with constraints [4].

As future work, we will prove correctness and completeness for more general forms of multi-agent systems, where every agent can perform speculative computation. Our current framework is focused on master-slave multi-agent systems, and defines the operational model of master agents. To handle a more general multi-agent system, we need to guarantee the appropriate computation of the overall system by additionally considering communication paths among agents. As another direction, we will also consider applications for this framework.

References

1. J. Jaffar, M. J. Maher, K. Marriott, and P. J. Stuckey. The semantics of constraint logic programs. *Journal of Logic Programming*, 37(1-3):1–46, 1998.
2. S. Janson and S. Haridi. Programming paradigms of the andorra kernel language. In *Proc. of ISLP'91*, pages 167–186, 1991.
3. R. A. Kowalski and F. Sadri. From logic programming towards multi-agent systems. In *Annals of Mathematics and Artificial Intelligence*, volume 25, pages 391–419, 1999.
4. K. Satoh, P. Codognet, and H. Hosobe. Speculative constraint processing in multi-agent systems. In *Proc. of PRIMA2003, LNCS 2891*, pages 133 – 144, 2003.
5. K. Satoh, K. Inoue, K. Iwanuma, and C. Sakama. Speculative computation by abduction under incomplete communication environments. In *Proc. of ICMAS2000*, pages 263–270, 2000.
6. K. Satoh and K. Yamamoto. Speculative computation with multi-agent belief revision. In *Proc. of AAMAS2002*, pages 897 – 904, 2002.
7. C. Schulte. Programming constraint services: High-level programming of standard and new constraint services. In *LNCS*, volume 2302. Springer Verlag, 2002.

Constitutive Norms in the Design of Normative Multiagent Systems

Guido Boella[1] and Leendert van der Torre[2]

[1]Dipartimento di Informatica - Università di Torino - Italy. email: guido@di.unito.it
[2]CWI Amsterdam and TU Delft - The Netherlands. email: torre@cwi.nl

Abstract. In this paper, we consider the design of normative multiagent systems composed of both constitutive and regulative norms. We analyze the properties of constitutive norms, in particular their lack of reflexivity, and the trade-off between constitutive and regulative norms in the design of normative systems. As methodology we use the metaphor of describing social entities as agents and of attributing them mental attitudes. In this agent metaphor, regulative norms expressing obligations and permissions are modelled as goals of social entities, and constitutive norms expressing "counts-as" relations are their beliefs.

1 Introduction

Legal systems are often modelled using regulative norms, like obligations and permissions [1]. However, a large part of the legal code does not contain prohibitions and permissions, but definitions for classifying the commonsense world under legal categories, like contract, money, property, marriage. Regulative norms can refer to this legal classification of reality.

Consider the consequences for the design of legal systems. For example, in [2] we address the issue of designing obligations to achieve the objectives of the legal system. However, the problem has not been studied of how to design legal systems composed of both constitutive and regulative norms. For modelling constitutive norms, specialized formalisms for counts-as conditionals have been introduced [3–5], but it remains unclear how to relate them to regulative norms. In contrast, as Artosi *et al.* [3] argue, for constitutive norms to be norms it is necessary that "their conditional nature exhibits some basic properties enjoyed by the usual normative links".

Obligations, prohibitions and permissions have a conditional nature. Their conditions could directly refer to entities and facts of the commonsense world, but they can rather refer to a legal and more abstract classification of the world, making them more independent from the commonsense view. E.g., they refer to money instead of paper sheets, to properties instead of houses and fields. This more natural and economical way to model the relation between commonsense reality and legal reality uses "count-as" conditionals, and allows regulative norms to refer to the legal classification of reality. In this way, e.g., it is not necessary that each regulative norm refers to all the conditions involved in the classification of paper as money or of houses and fields as properties. Moreover, it is not necessary that regulative norms manage the exceptions in the classification, e.g., that a fake bill is not money or that some field is not considered as a property. Finally, by referring to the legal classification of reality only, regulative norms are not sensitive anymore to changes in the classification: a new bill can be introduced without changing the regulative norms concerning money, or a new form of property or a new kind of marriage can be introduced without changing the relevant norms.

However, the trade-off and equivalences between systems made purely of regulative norms and those including also constitutive norms cannot be easily captured by specialized formalisms. They either consider only regulative norms, such as deontic logic, or only constitutive norms, such as logics of counts-as conditionals, or, finally, with formalisms using very different formalizations for modelling the two kinds of norms.

In [6], to model social reality, we have introduced constitutive norms in our normative multiagent systems. In this paper we use normative multiagent systems to model the design of legal systems. In particular, the research questions of this paper are: What properties have constitutive norms? In [6] we use rules satisfying the identity property, thus making the "counts-as" relation reflexive. This is a undesired property if constitutive norms provide a classification of reality in term of legal categories. In this paper we remedy this by modelling "counts-as" as input/output conditionals. This is an alternative solution with respect to the one proposed by Artosi *et al.* [3]. Secondly, how can regulative and constitutive norms be traded-off against each other in the design of legal systems? If we replace constitutive norms in a legal system with regulative ones, then we loose the abstraction provided by legal classification.

The main advantage of our approach in comparison with other accounts, is that we combine constitutive and regulative norms in a single conceptual model. As methodology we use our model of normative multiagent systems introduced in AI and agent theory to model social reality and agent organizations [7]. The basic assumptions of our model are that beliefs, goals and desires of an agent are represented by conditional rules, and that, when an agent takes a decision, it recursively models [8] the other agents interfering with it in order to predict their reaction to its decision as in a game. Most importantly, the normative system itself can be conceptualized as an agent with whom it is possible to play games to understand what will be its reaction to the agent's decision: to consider its behavior as a violation and to sanction it. In the model presented in [6], regulative norms are represented by the goals of the normative system and constitutive norms as its beliefs. In this paper we discuss how trade-off problem between constitutive and regulative norms can be handled by as the trade-off between beliefs and goals of the normative system. The cognitive motivations of the agent metaphor underlying our framework are discussed in [9].

The paper is organized as follows. In Section 2 we describe the agent metaphor. In Section 3 we introduce a logic which does not satisfy identity. In Section 4 we discuss the relation between constitutive and regulative norms. In Section 5 we introduce a formal model where we discuss the properties of constitutive norms and in Section 6 the trade-off with regulative ones. Comparison with related work and conclusion end the paper.

2 Attributing mental attitudes

We start with a well known definition:“ *Normative systems* are sets of agents (human or artificial) whose interactions can fruitfully be regarded as norm-governed; the norms prescribe how the agents ideally should and should not behave [...]. Importantly, the norms allow for the possibility that actual behaviour may at times deviate from the ideal, i.e. that violations of obligations, or of agents rights, may occur” [1].

This definition of Carmo and Jones does not seem to require that the normative system is autonomous, or that its behavior is driven by beliefs and desires.

In [6] we use the agent metaphor which attributes mental attitudes to normative systems in order to explain normative reasoning in autonomous agents. The normative system is considered as an agent with whom the bearer of the norms plays a game. Henceforth, we can call it the normative agent.

Our motivation for using the agent metaphor is inspired by the interpretation of normative *multiagent* systems as dynamic social orders. According to Castelfranchi [10], a social order is a pattern of interactions among interfering agents “such that it allows the satisfaction of the interests of some agent”. These interests can be a delegated goal, a value that is good for everybody or for most of the members; for example, the interest may be to avoid accidents. We say that agents attribute the mental attitude ‘goal’ to the normative system, because all or some of the agents have socially delegated goals to the normative system; these goals are the content of the obligations regulating it.

Moreover, social order requires *social control*, “an incessant local (micro) activity of its units” [10], aimed at restoring the regularities prescribed by norms. Thus, the agents attribute to the normative system, besides goals, also the ability to autonomously enforce the conformity of the agents to the norms, because a dynamic social order requires a continuous activity for ensuring that the normative system’s goals are achieved. To achieve the normative goal the normative system forms the subgoals to consider as a violation the behavior not conform to it and to sanction violations.

Searle argues that there are two types of norms: “Some rules regulate antecedently existing forms of behaviour. For example, the rules of polite table behaviour regulate eating, but eating exists independently of these rules. Some rules, on the other hand, do not merely regulate an antecedently existing activity called playing chess; they, as it were, create the possibility of or define that activity. The activity of playing chess is constituted by action in accordance with these rules. Chess has no existence apart from these rules. The institutions of marriage, money, and promising are like the institutions of baseball and chess in that they are systems of such constitutive rules or conventions” ([11], p. 131).

According to Searle, institutional facts like marriage, money and private property emerge from an independent ontology of “brute” natural facts through constitutive norms of the form “such and such an X counts as Y in context C” where X is any object satisfying certain conditions and Y is a label that qualifies X as being something of an entirely new sort. Examples of constitutive norms are “X counts as a presiding official in a wedding ceremony”, “this bit of paper counts as a five euro bill” and “this piece of land counts as somebody’s private property”.

In our model, we define constitutive norms in terms of the normative system’s belief rules and the institutional facts as the consequences of these beliefs rules.

The propositions describing the world are distinguished in two categories: first, what Searle calls "brute facts": natural facts and events produced by the actions of the agents. Second, "institutional facts": a legal classification of brute facts; they belong only to the beliefs of the normative system and have no direct counterpart in the world. Belief rules connect beliefs representing the state of the world to other beliefs which are their consequences. They have a conditional character and are represented in the same rule based formalism as goals and desires. In the case of the normative system the belief rules have as consequences not other beliefs about brute facts in the world (e.g., "if a glass drops, it breaks"), but new legal, institutional facts whose existence is related only to the normative system. These belief rules, moreover, can connect also institutional facts to other institutional facts.

This type of belief rules express the *counts-as* relations which are at the basis of constitutive norms. It is important that belief rules have a conditional character, since they must reflect the conditional nature of the counts-as relation as proposed by Searle: "such and such an X counts as Y in context C".

A fact p counts as an institutional fact q in context C for normative system $\mathbf{n}$ $counts\text{-}as_{\mathbf{n}}(p, q \mid C)$, iff agent $\mathbf{n}$ believes that $p \wedge C$ has q as a consequence.

We extend this approach advocated in [6] in two ways. First we give a logical analysis of counts-as, and we argue that it requires an identity free logic. Second we discuss the trade-off between the two kinds of norms.

3 Input/output logic

A disadvantage of the approach in [6] is that given the reflexivity of counts-as we have that "A counts as A", which is in contrast with our intuition and with other approaches (but see Section 7 for a discussion). In particular, since the counts-as relation classifies brute facts in legal categories, a brute fact A cannot be also a legal category: they are ontologically heterogeneous concepts, thus we keep them separate for the purpose of legal classification.

We therefore want to use an identity free logic, for which we take a simplified version of the input/output logics introduced in [12, 13]. In this section we explain how it works. A rule set is a set of ordered pairs $P \rightarrow q$, where P is a set of propositional variables and q a propositional variable. For each such pair, the body P is thought of as an input, representing some condition or situation, and the head q is thought of as an output, representing what the rule tells us to be believed, desirable, obligatory or whatever in that situation. Makinson and van der Torre write (P, q) to distinguish input/output rules from conditionals defined in other logics, to emphasize the property that input/output logic does not necessarily obey the identity rule. In this paper we do not follow this convention.

In this paper, input and output are respectively a set of literals and a literal. We use a simplified version of input/output logics, since it keeps the formal exposition simple and it is sufficient for our purposes here. In Makinson and van der Torre's input/output logics, the input and output can be arbitrary propositional formulas, not just sets of literals and literal as we do here. Consequently, in input/output logic there are additional rules for conjunction of outputs and for weakening outputs.

Definition 1 (Input/output logic).

Let X be a set of propositional variables, the set of literals built from X, written as Lit(X), is $X \cup \{\neg x \mid x \in X\}$, and the set of rules built from X, written as Rul$(X) = 2^{Lit(X)} \times$ Lit(X), is the set of pairs of a set of literals built from X and a literal built from X, written as $\{l_1, \ldots, l_n\} \to l$. We also write $l_1 \wedge \ldots \wedge l_n \to l$ and when $n = 0$ we write $\top \to l$. Moreover, for $x \in X$ we write $\sim x$ for $\neg x$ and $\sim(\neg x)$ for x.

Moreover, let Q be a set of pointers to rules and $MD : Q \to Rul(X)$ is a total function from the pointers to the set of rules built from X.

Let $S = MD(Q)$ be a set of rules $\{P_1 \to q_1, \ldots, P_n \to q_n\}$, and consider the following proof rules strengthening of the input (SI), disjunction of the input (OR), cumulative transitivity (CT) and Identity (Id) defined as follows:

$$\frac{p \to r}{p \wedge q \to r} SI \qquad \frac{p \wedge q \to r, p \wedge \neg q \to r}{p \to r} OR \qquad \frac{p \to q, p \wedge q \to r}{p \to r} CT \qquad \frac{}{p \to p} Id$$

The following output operators are defined as closure operators on the set S using the rules above.

out_1*: SI (simple-minded output)* out_3*: SI+CT (simple-minded reusable output)*
out_2*: SI+OR (basic output)* out_4*: SI+OR+CT (basic reusable output)*

Moreover, the following four throughput operators are defined as closure operators on the set S. out_i^+: out_i+Id (throughput) We write $out(Q)$ for any of these output operations and $out^+(Q)$ for any of these throughput operations. We also write $l \in out(Q, L)$ iff $L \to l \in out(Q)$, and $l \in out^+(Q, L)$ iff $L \to l \in out^+(Q)$.

Example 1. Given $MD(Q) = \{a \to x, x \to z\}$ the output of Q contains $x \wedge a \to z$ using the rule SI. Using also the CT rule, the output contains $a \to z$. $a \to a$ follows only if there is the Id rule.

A technical reason to distinguish pointers from rules is to facilitate the description of the priority ordering we introduce in the following definition.

The notorious contrary-to-duty paradoxes such as Chisholm's and Forrester's paradox have led to the use of constraints in input/output logics [13]. The strategy is to adapt a technique that is well known in the logic of belief change - cut back the set of norms to just below the threshold of making the current situation inconsistent.

In input/output logics under constraints, a set of mental attitudes and an input does not have a set of propositions as output, but a set of set of propositions. We can infer a set of propositions by for example taking the join (credulous) or meet (sceptical), or something more complicated. Besides, we can adopt an output constraint (the output has to be consistent) or an input/output constraint (the output has to be consistent with the input). In this paper we only consider the input/output constraints. The following definition is inspired by [14] where we extend constraints with priorities:

Definition 2 (Constraints).

Let $\geq$: $2^Q \times 2^Q$ be a transitive and reflexive partial relation on the powerset of the pointers to rules containing at least the subset relation. Moreover, let out be an input/output logic. We define:

– $maxfamily(Q, P)$ *is the set of $\subseteq$-maximal subsets Q' of Q such that $out(Q', P) \cup P$ is consistent.*

– *preffamily*$(Q, P, \geq)$ *is the set of* $\geq$*-maximal elements of preffamily*(Q, P).
– *outfamily*$(Q, P, \geq)$ *is the output under the elements of maxfamily, i.e.,* $\{out(Q', P) \mid Q' \in$ *preffamily*$(Q, P, \geq)\}$.
– $P \to x \in out_{\cup}(Q, \geq)$ *iff* $x \in \cup$*outfamily*$(Q, P, \geq)$
$P \to x \in out_{\cap}(Q, \geq)$ *iff* $x \in \cap$*outfamily*$(Q, P, \geq)$

In case of contrary to duty obligations, the input represents something which is inalterably true, and an agent has to ask himself which rules (output) this input gives rise to: even if the input should have not come true, an agent has to "make the best out of the sad circumstances" [15].

Example 2. Let $MD(\{a, b, c\}) = \{a = (\top \to m), b = (p \to n), c = (o \to \neg m)\}$, $\{b, c\} > \{a, b\} > \{a, c\}$, where by $A > B$ we mean as usual $A \geq B$ and $B \not\geq A$.
$maxfamily(Q, \{o\}) = \{\{a, b\}, \{b, c\}\}$,
preffamily$(Q, \{o\}, \geq) = \{\{b, c\}\}$,
outfamily$(Q, \{o\}, \geq) = \{\{\neg m\}\}$

The maxfamily includes the sets of applicable compatible pointers to rules together with all non applicable ones: e.g., the output of $\{a, c\}$ in the context $\{o\}$ is not consistent. Finally $\{a\}$ is not in maxfamily since it is not maximal, we can add the non applicable rule b. Then *preffamily* is the preferred set $\{b, c\}$ according to the ordering on set of rules above. The set *outfamily* is composed by the consequences of applying the rules $\{b, c\}$ which are applicable in o (c): $\neg m$.

Due to space limitations we have to be brief on details with respect to input/output logics, see [12, 13] for the semantics of input/output logics, further details on its proof theory, its possible translation to modal logic, alternative constraints, and examples.

4 Constitutive norms vs regulative norms

Why are constitutive norms needed in a normative system? In [6], we argue that, first, regulative norms are not categorical, but conditional: they specify all their applicability conditions. In case of complex and rapidly evolving systems new situations arise which should be considered in the conditions of the norms. Thus, new regulative norms must be introduced each time the applicability conditions must be extended to include new cases. In order to avoid changing existing norms or adding new ones, it would be more economic that regulative norms could factor out particular cases and refer, instead, to more abstract concepts only. Hence, the normative system should include some mechanism to introduce new institutional categories of abstract entities for classifying possible states of affairs. Norms could refer to this institutional classification of reality rather than to the commonsense classification: changes to the conditions of the norms would be reduced to changes to the institutional classification of reality. Second, the dynamics of the social order which the normative system aims to achieve is due to the evolution of the normative system over time, which introduces new norms, abrogates outdated ones, and, as just noticed, changes its institutional classification of reality. So the normative system must specify how the normative system itself can be changed by introducing new regulative norms and new institutional categories, and specify by whom the changes can be done.

In this paper we discuss how constitutive norms, even if they can be replaced by regulative norms, allow to create a level of abstraction to which regulative norms can refer to, making to less sensitive to the changes in the legal system. The cons of introducing constitutive norms is that new rules are necessary, so that a trade-off must be found between the need of abstraction and the complexity of the normative system.

As a running example, consider a society where the fact that a field has been fenced by an agent counts as the fact that the field is property of that agent. In our model this relation is expressed as a belief attributed to the normative system. The fence is a physical "brute" fact, while the fact that it is a property of someone is only an institutional fact attributed to the beliefs of the normative system.

Assume now that the normative system has as goals that if a field is fenced, no one enters it and that if a fenced field is entered, this action is considered as a violation and the violation is sanctioned. These goals form an obligation not to trespass a fenced field. However, the same legal system could have been designed in a different way using the constitutive norm above: a fenced field counts as property. The constitutive norm introduces the legal category of property which an obligation not to trespass a property can refer to: it is obligatory not to trespass property. The two legal systems are equivalent in the sense that in the same situation, the same violations hold; on the other hand, they are different since the latter introduce a legal classification of reality; thus, the obligation has as condition the institutional fact that the field is a property: the field being a property is an institutional fact believed by the normative system, while entering the field is a brute fact.

Analogously, in the purely regulative legal system, a permission to enter a fenced field if it is close to a river could be added. This permission is an exception to the obligation not to trespass fenced fields. In the second legal system, the same purpose can be reached by adding a constitutive norm which states that a field close to the river, albeit fenced, is not a property. Note that this is different from saying that a field on the river is a property that can be trespassed, a fact which is expressed by a permission to enter a property close to the river.

The possibility that institutional facts appear as conditions in the goals of the normative system or as goals themselves explains the following puzzling assertion of Searle [16]: "constitutive rules constitute (and also regulate) an activity the existence of which is logically dependent on the rules" (p.34). In our model constitutive norms regulate a social activity since they create institutional facts that are conditions or objects of regulative norms. In our metaphorical mapping regulative norms are goals, and goals base their applicability in a certain situation on the beliefs of the agent; in the previous example, being a property indirectly regulates the behavior of agents, since entering a field is a violation only if it is a property; if a field is not a property, the goal of considering trespassing a violation does not apply.

Searle [16] interprets the creation institutional facts also in terms of what he calls "status functions": "the form of the assignment of the new status function can be represented by the formula 'X counts as Y in C'. This formula gives us a powerful tool for understanding the form of the creation of the institutional fact, because the form of the collective intentionality is to impose that status and its function, specified by the Y term, on some phenomenon named by the X term", p.46.

Where "the ascription of function ascribes *the use to which we intentionally put* these objects" (p.20). In our model, this teleological aspect of the notion of function depends on the fact that institutional facts make conditional goals relevant as they appear in the conditions of regulative norms or as goals themselves. The aim of fencing a field is to prevent trespassing: the obligation defines the function of property, since it is defined in terms of goals of the normative system. Hence, Searle's assertion that "the institutions [. . .] are systems of such constitutive rules" is partial: institutions are systems where constitutive (i.e., beliefs) and regulative (i.e., goals) rules interacts. In our model, they interplay in the same way as goals and beliefs do in agents.

5 The formal model

The definition of the agents is inspired by the rule based BOID architecture [17], though in our theory, and in contrast to the BOID architecture, obligations are not taken as primitive concepts. Beliefs, desires and goals are represented by conditional rules rather then in a modal framework. We use in our model only goals rather than intentions since we consider only on decision step instead of having plans for the future moves.

We assume that the base language contains boolean variables and logical connectives. The variables are either *decision variables* of an agent, which represent the agent's actions and whose truth value is directly determined by it, or *parameters*, which describe both the state of the world and *institutional facts*, and whose truth value can only be determined indirectly. Our terminology is borrowed from Lang *et al.* [18].

Given the same set of mental attitudes, agents reason and act differently: when facing a conflict among their motivations and beliefs, different agents prefer to fulfill different goals and desires. We express these agent characteristics by a priority relation on the mental attitudes which encode, as detailed in [17], how the agent resolves its conflicts. The priority relation is defined on the powerset of the mental attitudes such that a wide range of characteristics can be described, including social agents that take the desires or goals of other agents into account. The priority relation contains at least the subset-relation which expresses a kind of independence among the motivations.

Definition 3 (Agent set). *An agent set is a tuple $\langle A, X, B, D, G, AD, \geq\rangle$, where:*

– *the agents A, propositional variables X, agent beliefs B, desires D and goals G are five finite disjoint sets.*
– *B, D, G are sets of pointers to rules. We write $M = D \cup G$ for the motivations defined as the union of the desires and goals.*
– *an agent description $AD : A \rightarrow 2^{X \cup B \cup M}$ is a total function that maps each agent to sets of variables (its decision variables), beliefs, desires and goals, but that does not necessarily assign each variable to at least one agent. For each agent $b \in A$, we write X_b for $X \cap AD(b)$, and B_b for $B \cap AD(b)$, D_b for $D \cap AD(b)$, etc. We write parameters $P = X \setminus \cup_{b \in A} X_b$.*

– *a priority relation $\geq: A \rightarrow 2^{M \cup B} \times 2^{M \cup B}$ is a function from agents to a transitive and reflexive partial relation on the powerset of the motivations containing at least the subset relation. We write $\geq_b$ for $\geq(b)$.*
Since goals have priority over desires we have that given $S, S' \subseteq M$, for all $a \in A$, $S >_a S'$ if $S \setminus S' \subseteq G$ and $S' \setminus S \subseteq D$.

Example 3. $A = \{\mathbf{a}\}$, $X_{\mathbf{a}} = \{\textit{trespass}\}$, $P = \{s, \textit{fenced}\}$, $D_{\mathbf{a}} = \{d_1, d_2\}$, $\geq_{\mathbf{a}} = \{d_2\} \geq \{d_1\}$. There is a single agent, agent $\mathbf{a}$, who can trespass a field. Moreover, it can be sanctioned and the field can be fenced. It has two desires, one to trespass (d_1), another one not to be sanctioned (d_2). The second desire is more important.

In a multiagent system, beliefs, desires and goals are abstract concepts which are described by rules built from literals.

Definition 4 (Multiagent system). *A multiagent system is a tuple $\langle A, X, B, D, G, AD, MD, \geq\rangle$, where $\langle A, X, B, D, G, AD, \geq\rangle$ is an agent set, and the mental description $MD : (B \cup M) \rightarrow \textit{Rul}(X)$ is a total function from the sets of beliefs, desires and goals to the set of rules built from X. For a set of mental attitudes $S \subseteq B \cup M$, we write $MD(S) = \{MD(q) \mid q \in S\}$.*

Example 4 (Continued). $MD(d_1) = \top \rightarrow \textit{trespass}$, $MD(d_2) = \top \rightarrow \neg s$.

In the description of the normative system, we do not introduce norms explicitly, but we represent several concepts which are illustrated in the following sections. Institutional facts (I) represent legal abstract categories which depend on the beliefs of the normative system and have no direct counterpart in the world. $F = X \setminus I$ are what Searle calls "brute facts": physical facts like the actions of the agents and their effects. $V(x, \mathbf{a})$ represents the decision of agent $\mathbf{n}$ that recognizes x as a violation by agent $\mathbf{a}$. The goal distribution $GD(\mathbf{a}) \subseteq G_{\mathbf{n}}$ represents the goals of agent $\mathbf{n}$ the agent $\mathbf{a}$ is responsible for.

Definition 5 (Normative system). *A normative multiagent system, written as $NMAS$, is a tuple $\langle A, X, B, D, G, AD, MD, \geq, \mathbf{n}, I, V, GD\rangle$ where the tuple $\langle A, X, B, D, G, AD, MD, \geq\rangle$ is a multiagent system, and*

– *the normative system $\mathbf{n} \in A$ is an agent.*
– *the institutional facts $I \subseteq P$ are a subset of the parameters.*
– *the norm description $V : \textit{Lit}(X) \times A \rightarrow X_{\mathbf{n}} \cup P$ is a function from the literals and the agents to the decision variables of the normative system and the parameters.*
– *the goal distribution $GD : A \rightarrow 2^{G_{\mathbf{n}}}$ is a function from the agents to the powerset of the goals of the normative system, such that if $L \rightarrow l \in MD(GD(\mathbf{a}))$, then $l \in \textit{Lit}(X_{\mathbf{a}} \cup P)$.*

Agent $\mathbf{n}$ is a normative system who has the goal that fenced fields are not trespassed.

Example 5 (Continued). There is agent $\mathbf{n}$, representing the normative system.
$X_{\mathbf{n}} = \{s, V(\textit{trespass}, \mathbf{a})\}$, $P = \{\textit{fenced}\}$, $D_{\mathbf{n}} = G_{\mathbf{n}} = \{g_1\}$, $MD(g_1) = \{\textit{fenced} \rightarrow \neg\textit{trespass}\}$, $GD(\mathbf{a}) = \{g_1\}$.

Agent $\mathbf{n}$ can sanction agent $\mathbf{a}$, because s is no longer a parameter but a decision variable. $V(\textit{trespass}, \mathbf{a})$ represents the fact that the normative system considers a violation the action of $\mathbf{a}$ trespassing the field. It has the goal that fenced fields are not trespassed, and it has distributed this goal to agent $\mathbf{a}$.

In the following , we use an input/output logic out to define whether a desire or goal implies another one and to define the application of a set of belief rules to a set of literals; in both cases we use the out_3 operation since it has the desired logical property of not satisfying identity.

Regulative norms are conditional obligations with an associated sanction and conditional permissions. The definition of obligation contains several clauses. The first and central clause of our definition defines obligations of agents as goals of the normative system, following the 'your wish is my command' metaphor. It says that the obligation is implied by the desires of the normative system **n**, implied by the goals of agent **n**, and it has been distributed by agent **n** to the agent. The latter two steps are represented by $out(GD(\mathbf{a}), \geq_{\mathbf{n}})$.

The second and third clause can be read as "the absence of p is considered as a violation". The association of obligations with violations is inspired by Anderson's reduction of deontic logic to alethic logic [19]. The third clause says that the agent desires that there are no violations, which is stronger than that it does not desire violations, as would be expressed by $\top \to V(\sim x, a) \notin out(D_{\mathbf{n}}, \geq_{\mathbf{n}})$.

The fourth and fifth clause relate violations to sanctions. The fourth clause says that the normative system is motivated not to count behavior as a violation and apply sanctions as long as their is no violation, because otherwise the norm would have no effect. Finally, for the same reason the last clause says that the agent does not like the sanction. The second and fourth clauses can be considered as instrumental norms [20] contributing to the achievement of the main goal of the norm.

Definition 6 (Obligation). *Let $NMAS = \langle A, X, B, D, G, AD, MD, \geq, \mathbf{n}, I, V, GD\rangle$ be a normative multiagent system. Agent $\mathbf{a} \in A$ is obliged to see to it that $x \in$ Lit$(X_{\mathbf{a}} \cup P)$ with sanction $s \in$ Lit$(X_{\mathbf{n}} \cup P)$ in context $Y \subseteq$ Lit(X) in $NMAS$, written as $NMAS \models O_{\mathbf{an}}(x, s|Y)$, if and only if:*

1. $Y \to x \in out(D_{\mathbf{n}}, \geq_{\mathbf{n}}) \cap out(GD(\mathbf{a}), \geq_{\mathbf{n}})$*: if Y then agent $\mathbf{n}$ desires and has as a goal that x, and this goal has been distributed to agent $\mathbf{a}$.*
2. $Y \cup \{\sim x\} \to V(\sim x, \mathbf{a}) \in out(D_{\mathbf{n}}, \geq_{\mathbf{n}}) \cap out(G_{\mathbf{n}}, \geq_{\mathbf{n}})$*: if Y and $\sim x$, then agent $\mathbf{n}$ has the goal and the desire $V(\sim x, \mathbf{a})$: to recognize it as a violation by agent $\mathbf{a}$.*
3. $\top \to \neg V(\sim x, \mathbf{a}) \in out(D_{\mathbf{n}}, \geq_{\mathbf{n}})$*: agent $\mathbf{n}$ desires that there are no violations.*
4. $Y \cup \{V(\sim x, \mathbf{a})\} \to s \in out(D_{\mathbf{n}}, \geq_{\mathbf{n}}) \cap out(G_{\mathbf{n}})$*: if Y and agent $\mathbf{n}$ decides $V(\sim x, \mathbf{a})$, then agent $\mathbf{n}$ desires and has as a goal that it sanctions agent $\mathbf{a}$.*
5. $Y \to \sim s \in out(D_{\mathbf{n}}, \geq_{\mathbf{n}})$*: if Y, then agent $\mathbf{n}$ desires not to sanction. This desire of the normative system expresses that it only sanctions in case of violation.*
6. $Y \to \sim s \in out(D_{\mathbf{a}}, \geq_{\mathbf{a}})$*: if Y, then agent $\mathbf{a}$ desires $\sim s$, which expresses that it does not like to be sanctioned.*

The rules in the definition of obligation are only motivations, and not beliefs, because a normative system may not recognize that a violation counts as such, or that it does not sanction it: it is up to its decision. Both the recognition of the violation and the application of the sanction are the result of autonomous decisions of the normative system that is modelled as an agent.

The beliefs, desires and goals of the normative agent - defining the obligations - are not private mental states of an agent. Rather they are collectively attributed by the agents of the normative system to the normative agent: they have a public character, and, thus, which are the obligations of the normative system is a public information.

Since conditions of obligations are sets of decision variables and parameters, institutional facts can be among them. In this way it is possible that regulative norms refer to institutional abstractions of the reality rather than to physical facts only.

Example 6 (Continued). Let: $\{g_1, g_2, g_4\} = G_{\mathbf{n}}$, $G_{\mathbf{n}} \cup \{g_3, g_5\} = D_{\mathbf{n}}$, $\{g_1\} = GD(\mathbf{a})$

$MD(g_2) = \{\mathit{fenced}, \mathit{trespass}\} \rightarrow V(\mathit{trespass}, \mathbf{a})$ $MD(g_3) = \top \rightarrow \neg V(\mathit{trespass}, \mathbf{a})$
$MD(g_4) = \{\mathit{fenced}, V(\mathit{trespass}, \mathbf{a})\} \rightarrow s$ $MD(g_5) = \mathit{fenced} \rightarrow \sim s$

$NMAS \models O_{\mathbf{an}}(\neg \mathit{trespass}, s \mid \mathit{fenced})$, since:

1. $\mathit{fenced} \rightarrow \mathit{trespass} \in out(D_{\mathbf{n}}, \geq_{\mathbf{n}}) \cap out(GD(\mathbf{a}), \geq_{\mathbf{n}})$
2. $\{\mathit{fenced}, \mathit{trespass}\} \rightarrow V(\mathit{trespass}, \mathbf{a}) \in out(D_{\mathbf{n}}, \geq_{\mathbf{n}}) \cap out(G_{\mathbf{n}}, \geq_{\mathbf{n}})$
3. $\top \rightarrow \neg V(\mathit{trespass}, \mathbf{a}) \in out(D_{\mathbf{n}}, \geq_{\mathbf{n}})$
4. $\{\mathit{fenced}, V(\mathit{trespass}, \mathbf{a})\} \rightarrow s \in out(D_{\mathbf{n}}, \geq_{\mathbf{n}}) \cap out(G_{\mathbf{n}}, \geq_{\mathbf{n}})$
5. $\mathit{fenced} \rightarrow \sim s \in out(D_{\mathbf{n}}, \geq_{\mathbf{n}})$
6. $\mathit{fenced} \rightarrow \sim s \in out(D_{\mathbf{a}}, \geq_{\mathbf{a}})$

[14], and can be overridden by obligations in turn. A permission to do x is an exception to an obligation not to do x if agent $\mathbf{n}$ has the goal that x is not considered as a violation under some condition. The permission overrides the prohibition if the goal that something does not count as a violation ($Y \wedge x \rightarrow \neg V(x, \mathbf{a})$) has higher priority in the ordering $\geq_{\mathbf{n}}$ on goal and desire rules with respect to the goal of a corresponding prohibition that x is considered as a violation ($Y' \wedge x \rightarrow V(x, \mathbf{a})$):

Definition 7 (Permission). *Agent $\mathbf{a} \in A$ is permitted by agent $\mathbf{n}$ to see to it that $x \in Lit(X_{\mathbf{a}} \cup P)$ under condition $Y \subseteq Lit(X)$, written as $NMAS \models P_{\mathbf{an}}(x \mid Y)$, iff $Y \cup \{x\} \rightarrow \neg V(x, \mathbf{a}) \in out(G_{\mathbf{n}}, \geq_{\mathbf{n}})$: if Y and x then agent $\mathbf{n}$ wants that x is not considered a violation by agent $\mathbf{a}$.*

Example 7 (Continued). Let $P = \{\mathit{fenced}, \mathit{river}\}$, $\{g_6\} > \{g_2\}$,
$MD(g_6) = \{\mathit{fenced}, \mathit{river}, \mathit{trespass}\} \rightarrow \sim V(\mathit{trespass}, \mathbf{a})$
Then $\{\mathit{fenced}, \mathit{river}, \mathit{trespass}\} \rightarrow \sim V(\mathit{trespass}, \mathbf{a}) \in out(D_{\mathbf{n}}, \geq_{\mathbf{n}}) \cap out(G_{\mathbf{n}}, \geq_{\mathbf{n}})$
Hence, $NMAS \models P_{\mathbf{an}}(\mathit{trespass} \mid \mathit{fenced} \wedge \mathit{river})$

Constitutive norms introduce new abstract categories of existing facts and entities, called institutional facts. We formalize the counts-as conditional as a belief rule of the normative system $\mathbf{n}$. Since the condition x of the belief rule is a variable it can be an action of an agent, a brute fact or an institutional fact. So, the counts-as relation can be iteratively applied.

Definition 8 (Counts-as relation). *Let $NMAS$=$\langle A, X, B, D, G, AD, MD, \geq, \mathbf{n}, I, V, GD\rangle$ be a normative multiagent system. A literal $x \in$ Lit(X) counts-as $y \in$ Lit(I) in context $C \subseteq$ Lit(X), $NMAS \models$ counts-as$_{\mathbf{n}}(x, y|C)$, iff $C \cup \{x\} \rightarrow y \in out(B_{\mathbf{n}}, \geq_{\mathbf{n}})$: if agent $\mathbf{n}$ believes C and x then it believes y.*

Example 8. $P \setminus I = \{$*fenced*$\}$, $I = \{$*property*$\}$, $X_{\mathbf{a}} = \{$*trespass*$\}$, $B'_{\mathbf{n}} = \{b'_1\}$, $MD(b'_1) =$ *fenced* $\rightarrow$ *property*

Consequently, $NMAS \models$ *counts-as*$_{\mathbf{n}}$(*fenced*, *property*$|\top$). This formalizes that for the normative system a fenced field counts as the fact that the field is a property of that agent. The presence of the fence is a physical "brute" fact, while being a property is an institutional fact. In situation $S = \{$*fenced*$\}$, given $B'_{\mathbf{n}}$ we have that the consequences of the constitutive norms are $out(B'_{\mathbf{n}}, S, \geq_{\mathbf{n}}) = \{$*property*$\}$

As shown in the example, the logic of constitutive norms does not satisfy identity: *fenced* is not a consequence, since it represents a brute fact and not an institutional fact. Constitutive norms, in contrast, provide a legal classification of reality in terms of institutional facts only.

The institutional facts can appear in the conditions of regulative norms as the following example shows.

Example 9 (Continued). A regulative norm which forbids trespassing can refer to the abstract concept of property rather than to fenced fields: $O_{\mathbf{an}}(\neg$*trespass*$, s \mid$ *property*$)$.
Let: $\{g'_1, g'_2, g'_4\} = G'_{\mathbf{n}}$, $G'_{\mathbf{n}} \cup \{g'_3, g'_5\} = D'_{\mathbf{n}}$, $\{g'_1\} = GD(\mathbf{a})$
$MD(g'_1) =$ *property* $\rightarrow \neg$*trespass* $MD(g'_2) = \{$*property*, *trespass*$\} \rightarrow V($*trespass*$, \mathbf{a})$
$MD(g'_3) = \top \rightarrow \neg V($*trespass*$, \mathbf{a})$ $MD(g'_4) = \{$*property*$, V($*trespass*$, \mathbf{a})\} \rightarrow s$
$MD(g'_5) =$ *property* $\rightarrow \sim s$
Then:

1. *property* $\rightarrow \neg$*trespass* $\in out(D_{\mathbf{n}}, \geq_{\mathbf{n}}) \cap out(GD(\mathbf{a}), \geq_{\mathbf{n}})$
2. $\{$*property*, *trespass*$\} \rightarrow V($*trespass*$, \mathbf{a}) \in out(D_{\mathbf{n}}, \geq_{\mathbf{n}}) \cap out(G_{\mathbf{n}}, \geq_{\mathbf{n}})$
3. $\top \rightarrow \neg V($*trespass*$, \mathbf{a}) \in out(D_{\mathbf{n}}, \geq_{\mathbf{n}})$
4. $\{$*property*$, V($*trespass*$, \mathbf{a})\} \rightarrow s \in out(D_{\mathbf{n}}, \geq_{\mathbf{n}}) \cap out(G_{\mathbf{n}}, \geq_{\mathbf{n}})$
5. *property* $\rightarrow \sim s \in out(D_{\mathbf{n}}, \geq_{\mathbf{n}})$
6. *property* $\rightarrow \sim s \in out(D_{\mathbf{a}}, \geq_{\mathbf{a}})$

As the system evolves, new cases can be added to the notion of property by means of new constitutive norms, without changing the regulative norms about property. E.g., if a field is inherited, then it is property of the heir: *inherit* $\rightarrow$ *property* $\in MD(B_{\mathbf{n}})$.

From a knowledge representation point of view, constitutive norms behave as *data abstraction* in programming languages: types are gathered in new abstract data types; new procedures are defined on the abstract data types to manipulate them. So it is possible to change the implementation of the abstract data type without modifying the programs using those procedures. In our case, it is possible to change the constitutive norms defining the institutional facts without modifying the regulative norms which refer to those institutional facts.

Since counts-as rules are beliefs and the logic is non-monotonic due to the priority ordering on the beliefs, counts-as can be used to express exceptions to the classification. This defeasible aspect mirrors the relation between obligations and permissions as exceptions [2].

6 The trade-off between constitutive and regulative norms

In this section, we extend our scenario described in Example 8-9 to design a legal system equivalent to the one of Example 6-7.

Example 10 (Continued). $B'_{\mathbf{n}} = \{b'_2\}$, $\{b'_2\} > \{b'_1\}$,
$MD(b'_2) = \mathit{fenced} \wedge \mathit{river} \rightarrow \neg \mathit{property}$.
$out(B'_{\mathbf{n}} = \{b'_1, b'_2\}, \geq_{\mathbf{n}}) = \{\{\mathit{fenced} \wedge \mathit{river} \rightarrow \neg \mathit{property}\}\}$ since
$maxfamily(B'_{\mathbf{n}}, S = \{\mathit{fenced}, \mathit{river}\}) = \{\{b'_1\}, \{b'_2\}\}$,
$\mathit{preffamily}(B'_{\mathbf{n}}, S = \{\mathit{fenced}, \mathit{river}\}, \geq_{\mathbf{n}}) = \{\{b'_2\}\}$,
$\mathit{outfamily}(B'_{\mathbf{n}}, S = \{\mathit{fenced}, \mathit{river}\}, \geq_{\mathbf{n}}) = \{\{\neg \mathit{property}\}\}$

Thus, $NMAS \models \mathit{counts\text{-}as}_{\mathbf{n}}(\mathit{fenced}, \neg \mathit{property} \mid \mathit{river})$ and this belief overrides the former one behind $\mathit{counts\text{-}as}_{\mathbf{n}}(\mathit{fenced}, \mathit{property} \mid \top)$. This formalizes that the normative system does not consider as a property a fenced field if it is close to a river.

We show how a system containing constitutive and regulative norms like in Example 8-10 can be interchanged with an equivalent system of regulative norms only like the one of Example 6-7. By equivalence we mean that in the same state of the world the same violations hold. Since it is possible to replace constitutive norms with regulative norms only, a trade-off can be found between adding constitutive norms and achieving a sufficient level of abstraction.

Even if input/output logic is an inference system on rules we cannot directly prove the equivalence on the rules defining regulative and constitutive norms since they refer to different sets of rules: goal rules and belief rules. We provide the equivalence in an indirect way by considering the combined output of the rules.

Given the operation out, we define a combined output relation: $output(Q, Z, S, \geq_{\mathbf{n}}) = out(Z, out(Q, S, \geq_{\mathbf{n}}) \cup S, \geq_{\mathbf{n}})$ where $Q \subseteq B_{\mathbf{n}}$, $Z \subseteq M_{\mathbf{n}}$ and $S \subset Lit(X \setminus I)$. The institutional facts are the result of the reasoning of the normative system, so they cannot be present in the initial state composed of brute facts.

Note that we reintroduce the brute facts S as the input of the output operation on the motivations Z since the output operation on beliefs does not satisfy identity. We need S since the conditions of regulative norms can refer to brute facts as well as to the institutional facts which are the consequences of the constitutive norms. In this way we distinguish between the legal classification of reality and the information concerning commonsense, among which the brute facts which are the input to constitutive norms. Even if we attribute belief rules to the normative system these must be distinguished from the belief rules of agents: these belief rules concern the relation between brute facts and constitute their commonsense view of the work. The normative system as agent, in contrast, does not contain any knowledge of this kind. The relevant commonsense inferences are performed by the real agents playing roles in the normative system.

In our examples we have:$output(B_{\mathbf{n}}, G_{\mathbf{n}}, S, \geq_{\mathbf{n}}) = output(B'_{\mathbf{n}}, G'_{\mathbf{n}}, S, \geq_{\mathbf{n}})$ for any $S \in Lit(X \setminus I)$.

Sketch of proof. We consider only the cases where the conditions of the goals and beliefs are satisfied. First, the normative system made of regulative norms only:

$output(B_{\mathbf{n}}, G_{\mathbf{n}}, S = \{\mathit{fenced}, \mathit{trespass}\}, \geq_{\mathbf{n}}) = out(G_{\mathbf{n}}, out(B_{\mathbf{n}}, S, \geq_{\mathbf{n}}) \cup S, \geq_{\mathbf{n}}) =$
$\{\neg \mathit{trespass}, V(\mathit{trespass}, \mathbf{a}), s\}$
from g_1, g_2, g_4, where $out(B_{\mathbf{n}}, S, \geq_{\mathbf{n}}) = \emptyset$ since $B_{\mathbf{n}} = \emptyset$.

In contrast:

$output(B_{\mathbf{n}}, G_{\mathbf{n}}, S{=}\{\textit{fenced}, \textit{river}, \textit{trespass}\}, \geq_{\mathbf{n}}){=}out(G_{\mathbf{n}}, out(B_{\mathbf{n}}, S, \geq_{\mathbf{n}}) \cup S, \geq_{\mathbf{n}})$
$= \{\neg \textit{trespass}, \neg V(\textit{trespass}, \mathbf{a}), {\sim} s\}$
(from g_1, g_5, g_6) where again $out(B_{\mathbf{n}}, S, \geq_{\mathbf{n}}) = \emptyset$.

In case of the legal system of Example 8 made of both constitutive and regulative norms:

$output(B'_{\mathbf{n}}, G'_{\mathbf{n}}, S{=}\{\textit{fenced}, \textit{trespass}\}, \geq_{\mathbf{n}}){=}out(G'_{\mathbf{n}}, out(B'_{\mathbf{n}}, S, \geq_{\mathbf{n}}) \cup S, \geq_{\mathbf{n}}) =$
$\{\neg \textit{trespass}, V(\textit{trespass}, \mathbf{a}), s\}$
(from g'_1, g'_2, g'_4) where $out(B'_{\mathbf{n}}, S, \geq_{\mathbf{n}}) = \{\textit{property}\}$ (from b'_1).

In contrast:

$output(B'_{\mathbf{n}}, G'_{\mathbf{n}}, S{=}\{\textit{fenced}, \textit{river}, \textit{trespass}\}, \geq_{\mathbf{n}}){=}out(G'_{\mathbf{n}}, out(B'_{\mathbf{n}}, S, \geq_{\mathbf{n}}){\cup}S, \geq_{\mathbf{n}}){=}$
$\{\neg \textit{trespass}, \neg V(\textit{trespass}, \mathbf{a}), {\sim} s\}$
(from g'_1, g'_3, g'_5) where $out(B'_{\mathbf{n}}, S, \geq_{\mathbf{n}}) = \{\neg \textit{property}\}$ (from b'_2).

7 Related work

While the formalization of regulative norms, like obligations, prohibitions and permissions, is often based in deontic logic on modal operators representing what is obligatory, forbidden or permitted, the formalization of constitutive norms is rather different. An attempt to make the notion of constitutive norm more precise is Jones and Sergot [5]'s formalization of the counts-as relation. For Jones and Sergot, the counts-as relation expresses the fact that a state of affairs or an action of an agent "is a sufficient condition to guarantee that the institution creates some (usually normative) state of affairs". As Jones and Sergot suggest, this relation can be considered as "constraints of (operative in) [an] institution", and they express these constraints as conditionals embedded in a modal operator. Jones and Sergot formalize this introducing a conditional connective $\Rightarrow_s$ to express the "counts-as" connection holding in the context of an institution s. They characterise the logic for $\Rightarrow_s$ as a classical conditional logic plus the axioms:

$((A \Rightarrow_s B) \wedge (A \Rightarrow_s C)) \supset (A \Rightarrow_s (B \wedge C))$
$((A \Rightarrow_s B) \wedge (C \Rightarrow_s B)) \supset ((A \vee C) \Rightarrow_s B)$
$((A \Rightarrow_s B) \wedge (B \Rightarrow_s C)) \supset (A \Rightarrow_s C)$

In addition, Jones and Sergot's analysis is integrated by introducing the normal KD modality D_s such that $D_s A$ means that A is "recognised by the institution s". Accordingly, it is adopted the schema: $(A \Rightarrow_s B) \supset D_s(A \supset B)$.

The limitation of this approach, according to Gelati *et al.* [21], is that the consequences of counts-as connections follow non-defeasibly (via the closure of the logic for modality D_s under logical implication), whereas defeasibility seems a key feature of such connections. The classical example is that in an auction if a person raises one hand, this may count as making a bid. However, this does not hold if he raises his hand and scratches his own head.

Finally, the adoption of the transitivity for their logic is criticized by Artosi *et al.* [3]. Artosi *et al.* [3]'s characterisation of the counts-as adopts a different perspective. Rather than introducing a logic for the counts-as connection, and then linking it with a D_s logic, they use one conditional operator $\Rightarrow$ to express any defeasible normative connections in any institutions. They use the same D_s operator as in [5] but they apply it to the components of normative links, to relativise them to a particular institution. Any institution can only state what normative situation holds for itself, given certain conditions, but according to a general type of conditionality. On the basis of $\Rightarrow$ they define a relativised $\Rightarrow_s$ operator: $(A \Rightarrow_s B) =_{def} (A \Rightarrow D_s B) \wedge (D_s A \Rightarrow D_s B)$

The connective $\Rightarrow$ is characterised by reflexivity and cumulative transitivity, whose combination does not prevent defeasibility. The system is completed by introducing a restricted version of the detachment of the consequent. To avoid losing non-monotonicity, Artosi *et al.* [3] do not accept the strengthening of antecedent property (SI in our input/output logic), thus making their logic weaker.

In contrast, in our model we accept the strengthening of antecedent (SI) rule and the cumulative transitivity (CT). We do not accept instead identity (Id). First of all, the adoption also of Id would make the system accepting also full transitivity. Non-monotonicity is achieved via the constraint mechanism which uses also a priority ordering on the mental attitudes. Secondly, we do not accept Id because we want to keep separate brute facts and institutional facts "whose nature - as also Artosi *et al.* [3] accept - is conceptually distinct from that of the empirical facts".

Finally, a recent interesting approach to counts-as is the model of Grossi *et al.* [4]. They propose a framework for grounding a new formal semantics of expressions such as: "A counts as B in institution c", or "B supervenes A in institution c". Constitutive norms are interpreted essentially as contextualized subsumption relations establishing taxonomies which hold only with respect to a specific (institutional) context.

8 Conclusions

In this paper we discuss the design of legal systems composed of constitutive and regulative norms. We model legal systems as normative multiagent systems where the normative system is modelled as an agent using the agent metaphor: constitutive norms are defined by the beliefs of the normative system and the regulative norms by its goals. The characteristic of the counts-as relation is that it is not reflexive. The trade-off problem between constitutive and regulative norms can be handled by as the trade-off between beliefs and goals of the normative system. We show that constitutive norms, even if they can be replaced by regulative norms, allow to create a level of abstraction to which regulative norms can refer to, making it less sensitive to the changes in the legal system.

In [6] we extend this framework to model the problem of how the normative system itself specifies who can change the normative system. This specification is made by means of constitutive norms describe what facts count as the creation of new regulative and constitutive norms in the normative system. This work is at the basis of the definition of contracts we make in [7]. Future work is, for example, elaborating the notion of context to study which properties hold for it, and introducing hierarchies of normative systems composed of both constitutive norms and regulative norms, as we do for obligations and permissions in [14].

References

1. Jones, A., Carmo, J.: Deontic logic and contrary-to-duties. In Gabbay, D., Guenthner, F., eds.: Handbook of Philosophical Logic. Kluwer (2001) 203–279
2. Boella, G., van der Torre, L.: Permissions and obligations in hierarchical normative systems. In: Procs. of ICAIL'03, ACM Press (2003) 109–118
3. Artosi, A., Rotolo, A., Vida, S.: On the logical nature of count-as conditionals. In: Procs. of LEA 2004 Workshop. (2004)
4. Grossi, D., Dignum, F., Meyer, J.J.: Contextual taxonomies. In: Procs. of Clima'04 Workshop. (2004)
5. Jones, A., Sergot, M.: A formal characterisation of institutionalised power. Journal of IGPL **3** (1996) 427–443
6. Boella, G., van der Torre, L.: Regulative and constitutive norms in normative multiagent systems. In: Procs. of KR'04, AAAI Press (2004) 255–265
7. Boella, G., van der Torre, L.: A game theoretic approach to contracts in multiagent systems. IEEE Transactions on Systems, Man and Cybernetics - Part C (2006)
8. Gmytrasiewicz, P.J., Durfee, E.H.: Formalization of recursive modeling. In: Procs. of ICMAS'95. (1995) 125–132
9. Boella, G., van der Torre, L.: From the theory of mind to the construction of social reality. In: Procs. of CogSci'05. (2005)
10. Castelfranchi, C.: Engineering social order. In: Procs. of ESAW'00, Berlin, Springer Verlag (2000) 1–18
11. Searle, J.: Speech Acts: an Essay in the Philosophy of Language. Cambridge University Press, Cambridge (UK) (1969)
12. Makinson, D., van der Torre, L.: Input-output logics. Journal of Philosophical Logic **29** (2000) 383–408
13. Makinson, D., van der Torre, L.: Constraints for input-output logics. Journal of Philosophical Logic **30(2)** (2001) 155–185
14. Boella, G., van der Torre, L.: Rational norm creation: Attributing mental attitudes to normative systems, part 2. In: Procs. of ICAIL'03, ACM Press (2003) 81–82
15. Hansson, B.: An analysis of some deontic logics. N$\hat{o}\hat{u}$s **3** (1969) 373–398
16. Searle, J.: The Construction of Social Reality. The Free Press, New York (1995)
17. Broersen, J., Dastani, M., Hulstijn, J., van der Torre, L.: Goal generation in the BOID architecture. Cognitive Science Quarterly **2(3-4)** (2002) 428–447
18. Lang, J., van der Torre, L., Weydert, E.: Utilitarian desires. Autonomous Agents and Multiagent Systems (2002) 329–363
19. Anderson, A.: The logic of norms. Logic et analyse **2** (1958)
20. Hart, H.: The Concept of Law. Clarendon Press, Oxford (1961)
21. Gelati, J., Governatori, G., Rotolo, N., Sartor, G.: Declarative power, representation, and mandate. A formal analysis. In: Procs. of JURIX 02, IOS press (2002)

Contextual Terminologies

Davide Grossi, Frank Dignum, John-Jules Ch. Meyer

Utrecht University,
The Netherlands
{davide,dignum,jj}@cs.uu.nl

Abstract. The paper addresses the issue of contextual representations of ontologies, as it arises in the area of normative system specifications for modeling multi-agent systems. To this aim, the paper proposes a formalization of a notion of contextual terminology, that is to say, a terminology holding only with respect to a specific context. The formalization is obtained by means of a formal semantics framework which enables the expressivity of common description logics to reason within contexts (intra-contextual reasoning), allowing at the same time the possibility to reason also about contexts and their interplay (inter-contextual reasoning). Using this framework, two complex scenarios are discussed in detail and formalized.

1 Introduction

The present research is motivated by problems stemming from the domain of normative system specifications for modeling multi-agent systems ([5, 19, 10]). In [6, 18] contexts have been advocated to play a central role in the specification of complex normative systems and normative systems of high complexity (for instance legal systems, or institutional ones) are viewed not only as regulative systems, but also as systems specifying conceptualizations, or categorizations, of the domain of entities they are supposed to regulate. In [12, 9] we proposed and applied a framework for representing this categorizing feature of normative systems via contextual taxonomic statements of the form "A counts as B in context C" ([16]), where concept descriptions A and B displayed a very simple logical form (essentially boolean compositions of concepts). This work intends to pursue that research line further adding the necessary expressivity (essentially the possibility to deal with attributes or roles, i.e., binary relations besides concepts) to model more complex scenarios: from simple taxonomies to rich description logic terminologies.

The final aim consists in obtaining a framework in which to represent ontologies of different contexts and to reason about them both in isolation, i.e., within the contexts (intra-contextual reasoning), and in interaction, i.e. between contexts (inter-contextual reasoning). For instance, at the intra-contextual level a typical question would be of the form: given a set of subsumption relations holding in context C, is A a subconcept of B in context C? At an inter-contextual level instead, a typical question would be: given that context C is more concrete than context D, is A a subconcept of B in context C? With such a machinery it would then be possible to represent the ontological aspect of the regulating activity of institutions in a formal way, and the ontologies of different institutions could then be rigorously specified and reasoned about. To do this, we show that the approach proposed in [12] can be naturally applied to richer description logic languages thus providing the necessary expressive power we are interested in. In fact, the framework presented here consists in a contextualized version of the semantics of description logics. The proposal is tested in detail against two different examples.

The exposition is structured according to the following outline. In Section 2 two scenarios are introduced which exemplify in detail the issues addressed here, and some preliminary considerations are drawn. Section 3 is dedicated to the exposition of the framework, and Section 4 to the formalization of the two scenarios introduced in Section 2. Conclusions follow.

2 Preliminaries

2.1 Scenarios

We now depict two scenarios in order to state, in clear terms, the kind of reasoning patterns we are aiming to capture formally. They exemplify quite typical forms of contextual conceptualizations occurring in the normative domain. The first scenario deals with a rule establishing sufficient conditions for a person to be liable of violating the regulation concerning access to public parks in three different municipalities. The second scenario deals with the refinement of a definition of "vehicle" from the abstract context of a general regulation to more concrete contexts of municipal regulations. From a logical point of view, they display description logic forms of reasoning at the level of the so-called taxonomical boxes (TBoxes)[1] (e.g., reasoning with value restriction and existential quantification, role subsumption) which were not yet available in our previous proposal ([12]).

Example 1. **(The public park scenario: "liability in parks")** In the regulation governing access to public parks in region R it is stated that vehicles are not allowed within public parks and that: "persons using vehicles within public parks are liable for violating the regulation". In this regulation no mention is made of (possible) subconcepts of the concept vehicle, e.g., cars, bicycles, which may help in identifying an instance of vehicle, nor it is stated what it actually means to drive a vehicle: does the fact that I am wheeling my bicycle imply that I am driving it? In municipal regulations subordinated to this regional one, and therefore inheriting its global directives, specific subconcepts are instead handled. In municipality M1 and M2 the following rule holds: "persons driving bicycles within parks are liable of violating the regulation". In M3 instead, it

[1] Taxonomical boxes or *terminologies* are, in the description logic vocabulary, sets of inclusion relations between concepts.

holds that to drive a bicycle does not constitute any violation. On the other hand, in all M1, M2 and M3 it holds that cars are not allowed in public parks. Moreover, in M2 it holds that "persons wheeling bicycles into public parks are not liable for violating the regulation" despite liability arises in case bicycles happen to be driven. In M1 and M3 instead, to wheel a bicycle is considered a way of driving it.

	DRIVE VEHICLE	DRIVE CAR	DRIVE BICYCLE	WHEEL BICYCLE
R	liable	*not classifiable*	*not classifiable*	*not classifiable*
M1	liable	liable	liable	liable
M2	liable	liable	liable	not liable
M3	liable	liable	not liable	not liable

Table 1. Liability in the public park scenario

In this scenario the concept of `vehicle` gets various interpretations. Instances of `car` (w.r.t. the terminologies presupposed by M1, M2 and M3) are always instances of `vehicle`, while instances of `bicycle` are only in some contexts also instances of `vehicle`. What also gets various interpretations is the relation driving: somehow driving in M2 has a different meaning than in M1 and M3. Table 1 displays how sufficient conditions for determining liability come down to be interpreted in three completely different ways by the contexts at issue, although in all contexts it holds that persons driving vehicles are to be considered liable. Note that context R cannot provide any qualification for actions such as driving or wheeling a bicycle simply because its language cannot express those notions.

Example 2. **(The public park scenario: "teenagers on skateboards")** Consider again a regulation governing access to public parks in region R where it is stated that: "vehicles are not allowed within public parks". Also in this regulation no mention is made of (possible) subconcepts of the concept vehicle. Nevertheless, a (partial) definition, specifying necessary conditions for something to be a vehicle, is stated: "vehicles are conveyances which transport persons or objects". In municipal regulations subordinated to this regional one subconcepts are instead introduced. This is done inheriting the definition stated at the R level and refining it either incrementing the number of necessary conditions for something to be considered a vehicle or stating sufficient ones. In municipality M1 the definition of vehicle is refined in the following sense: "self-propelled conveyances which transport persons or objects are vehicles" and "vehicles are self-propelled". In M2, instead, the definition of vehicle is simply closed without any refinement: "conveyances which transport persons or objects are vehicles". Besides, in both M1 and M2, it holds that"skateboards are conveyances which are not self-propelled" and "teenagers are persons". These rules determine a different behavior of M1 and M2 with respect to concepts such as "skateboards transporting teenagers". With respect to this concept the following rule holds in M1: "skateboards transporting teenagers are not vehicles". In M2 instead, it holds that: "skateboards transporting teenagers are vehicles".

The second scenario displays some other aspects of contextual conceptualizations. The concept of `vehicle` gets again various interpretations and is first specified in its necessary conditions by context R and then completely defined in the two concrete contexts M1 and M2. The abstract regulation states that all vehicles are conveyances transporting persons or objects, leaving thus open the possibility for some of such conveyances not to be vehicles. This is the case of skateboards in M1 since M1 refines the abstract rule establishing more necessary conditions (being self-propelled) for conveyances to be classified as vehicles. Context M2 instead, simply closes the abstract rule through establishing that being a conveyance transporting persons or objects is sufficient for being a vehicle. Because of this, the two contexts M1 and M2 validate terminologies diverging on the conceptualization of the complex concept "skateboards transporting teenagers".

These two scenarios exemplify interesting nuances typical of complex context-dependent conceptualizations[2]. We will constantly refer back to them in the remainder of the work, and our central aim will be to develop a formal semantics framework able to represent analogous scenarios and to provide thus a rigorous understanding of the forms of reasoning therein involved.

2.2 Contextualizing Terminologies

We want to devise a language and a semantics for talking about contextual terminologies. More in detail, this turns out to devise a formal morphology and a formal semantics meeting the following requirements.

Firstly, it should support reasoning about the validity of TBoxes with respect to contexts giving a semantics to expressions of the type: "the concept `bicycle` is a subconcept of the concept `vehicle` in context M1". Besides this, the framework should be able to express the fact that concepts may be unclassifiable within specific contexts, that is, that specific subsumptions cannot be said to be valid or not valid: in the context R of the regional regulation, whether a person wheeling a bicycle within a public park is to be considered liable of violating the regulation corresponds to a non evaluable subsumption since the concept at issue is not part of the language of the context R (see Figure 2.1). In some sense, it corresponds to a subsumption which is evaluated with respect to the wrong context. Therefore, we want the framework to be able to express whether a concept gets meaning within a context: "concept `bicycle` is meaningful with respect to context M1". Completely analogous expressions should be available in order to handle a contextualization of role (or attribute) hierarchies such as: "role wheel (wheeling) is a subrole of drive (driving) in context M2"

[2] It is instructive to notice that both scenarios represent instances of a typical form of contextual reasoning called "categorization" ([3]), or "perspective" ([1]), that is, the form of reasoning according to which a same set of entities is conceptualized in many different ways.

and "role wheel is meaningful in context M2". Secondly, it should provide a representation of context interplay. In particular, we will introduce: a *contextual disjunction* operator and a *contextual focus* operator[3].

The first one yields a union of contexts: the contexts "viruses" and "bacteria" can be unified on a language talking about microorganisms generating a more general context like "viral or bacterial microorganisms". The second one, which plays a central role in our framework, yields the context consisting of some information extracted from the context on which it is focused: the context categorizing "crocodiles", for instance, can be obtained via focusing the context which categorizes all reptiles on the language talking only about crocodiles and disregarding other reptiles. In other words, the operator prunes the information contained in the context "reptiles" focusing only on what is expressible in the language which talks about crocodiles and abstracting from the rest. Also *maximum* and *minimum* contexts will be introduced: these will represent the most general, and respectively the most specific, contexts on a language[4]. It is important to notice that all operations explicitly refer to a precise language on which the operation should take place. As we will see in the following section our formal language will be tuned to incorporate this feature.

Finally, our language should represent specific relations between contexts. Examples 1 and 2 consider groups of contexts in which all contexts are specializations of an abstract one (R). This suggests the consideration of a generality relation between contexts[5] expressing that a context is at most as general as another one: the context of the abstract regulation R is somehow more general than the concrete ones M1 and M2[6].

These intuitions about the semantics of context operators will be clarified and made more rigorous in Section 3.2 where the semantics of the framework will be presented, and in Section 4 where the examples will be formalized deploying all these types of expressions.

3 A Formal Framework

Our proposal consists in mixing the semantics of description logic ([2]) with the idea of modeling contexts as sets of models ([7]), delivering a framework able to represent reasoning about sets of concept subsumptions, i.e., taxonomical boxes (TBoxes), in a contextual setting.

[3] In [12, 11] the *focus* operation is called *abstraction*. We decided to modify our terminology in order to avoid confusions with other approaches to notions of abstraction like for instance [8].

[4] In this paper, we limit the number of context operations to disjunction and focus. More operations are formalized in [12]. It is worth noticing, in passing, that similar operations and special contexts are discussed in [17].

[5] Literature on context theory often addresses this type of relation between contexts. See for instance [15, 3].

[6] As the discussion of the formalization of the examples will show (Section 4), there are some more subtleties to be considered since R is not only more general but is also specified on a simpler language.

3.1 Language

The language we are defining can be seen as a meta-language for TBoxes defined on $\mathcal{AL}$ description logic languages, which handle also concept union, full existential quantification (we want to deal with concepts such as "either car or bicycle" and "persons who drive cars") and role complement (we want to be able to talk about roles such as "not driving")[7].

The alphabet of the language $\mathcal{L}^{CT}$ (*language for contextual terminologies*) contains therefore the alphabets of a family of languages $\{\mathcal{L}_i\}_{0\leq i\leq n}$. This family is built on the alphabet of a given "global" language $\mathcal{L}$ which contains all the terms occurring in the elements of the family. Moreover, we take $\{\mathcal{L}_i\}_{0\leq i\leq n}$ to be such that, for each non-empty subset of terms of the language $\mathcal{L}$, there exist a $\mathcal{L}_i$ which is built on that set and belongs to the family. Each $\mathcal{L}_i$ contains two non-empty finite sets $\mathbf{A_i}$ of atomic concepts (A), i.e., monadic predicates, and $\mathbf{R_i}$ of atomic roles (R), i.e., dyadic predicates. These languages contain also concepts and roles constructors. As to concept constructors, each $\mathcal{L}_i$ contains the zeroary operators $\bot$ (bottom concept) and $\top$ (top concept), the unary operator $\neg$ (complement), and the binary operators $\sqcap$ and $\sqcup$. As to role constructors, each $\mathcal{L}_i$ contains the unary operator $\overline{R}$ (role complement). Finally, the value restriction operator $\forall R.A$ ("the set of elements such that all elements that are in a relation R with them are instances of A") applies to role-concept pairs.

Besides, the alphabet of $\mathcal{L}^{CT}$ contains a finite set of context identifiers $\mathbf{c}$, two families of zeroary operators $\{\bot_i\}_{0\leq i\leq n}$ (minimum contexts) and $\{\top_i\}_{0\leq i\leq n}$ (maximum contexts), one family of unary operators $\{\mathrm{fcs}_i\}_{0\leq i\leq n}$ (contextual focus operator) , one family of binary operators $\{\curlyvee_i\}_{0\leq i\leq n}$ (contexts disjunction operator), one context relation symbol $\preccurlyeq$ (context c_1 "is less general than" context c_2), two meaningfulness relation symbols " $. \downarrow^c .$" (concept A is meaningful in context c) and " $. \downarrow^r .$" (role R is meaningful in context c), and finally two contextual subsumption relation symbols " $. : . \sqsubseteq^c .$" (within context c, concept A_1 is a subconcept of concept A_2) and " $. : . \sqsubseteq^r .$" (within context c, role R_1 is a subrole of role R_2) for, respectively, concept and role subsumption[8]. Lastly, the alphabet of $\mathcal{L}^{CT}$ contains also the sentential connectives $\sim$ (negation) and $\wedge$ (conjunction) [9].

Thus, the set Ξ of context constructs (ξ) is defined through the following BNF:

$$\xi ::= c \mid \bot_i \mid \top_i \mid \mathrm{fcs}_i\ \xi \mid \xi_1 \curlyvee_i \xi_2.$$

[7] This type of language is indeed an $\mathcal{ALC}$ conceptual language extended with role complement. See [2].

[8] We use superscripts here in order to distinguish between meaningfulness of concepts or roles, and subsumptions of concepts or roles. Nevertheless, in what follows, superscripts will be dropped when no confusion arises in order to lighten the notation.

[9] It might be worth remarking that language $\mathcal{L}^{CT}$ is, then, an expansion of each $\mathcal{L}_i$ language. Notice also that all operators on contexts are indexed with the language on which the operation they denote takes place.

Concept constructs and role constructs are defined in the standard way. The set P of roles descriptions (ρ) is defined through the following BNF:

$$\rho ::= R \mid \overline{\rho}.$$

The set Γ of concept descriptions (γ) is defined through the following BNF:

$$\gamma ::= A \mid \bot \mid \top \mid \neg\gamma \mid \gamma_1 \sqcap \gamma_2 \mid \forall\rho.\gamma.$$

Concept union and existential quantification are defined respectively as:

$$\gamma_1 \sqcup \gamma_2 =_{\text{def}} \neg(\neg\gamma_1 \sqcap \neg\gamma_2) \text{ and } \exists\rho.\gamma =_{\text{def}} \neg(\forall\rho.\neg\gamma).$$

Finally, the set $\mathcal{A}$ of assertions (α) is defined through the following BNF:

$$\alpha ::= \gamma \downarrow^c \xi \mid \rho \downarrow^r \xi \mid \xi : \gamma_1 \sqsubseteq^c \gamma_2 \mid \xi : \rho_1 \sqsubseteq^r \rho_2 \mid \xi_1 \preccurlyeq \xi_2 \mid \ \sim \alpha \mid \alpha_1 \wedge \alpha_2.$$

Strict contextual subsumption and contextual equivalence are obviously defined:

$$\begin{aligned}
&\xi : \gamma_1 \sqsubset^r \gamma_2 =_{\text{def}} \xi : \gamma_1 \sqsubseteq^c \gamma_2 \wedge \ \sim \xi : \gamma_2 \sqsubseteq^c \gamma_1\\
&\xi : \rho_1 \sqsubset^r \rho_2 =_{\text{def}} \xi : \rho_1 \sqsubseteq^r \rho_2 \wedge \ \sim \xi : \rho_2 \sqsubseteq^r \rho_1\\
&\xi : \gamma_1 \equiv^c \gamma_2 =_{\text{def}} \xi : \gamma_1 \sqsubseteq^c \gamma_2 \ \wedge \ \xi : \gamma_2 \sqsubseteq^c \gamma_1\\
&\xi : \rho_1 \equiv^r \rho_2 =_{\text{def}} \xi : \rho_1 \sqsubseteq^r \rho_2 \ \wedge \ \xi : \rho_2 \sqsubseteq^r \rho_1.
\end{aligned}$$

The set of atomic assertions of the language is then constituted by expressions enabling exactly the kind of expressivity required in Section 2.2: meaningfulness of concepts and roles in contexts, contextual subsumptions of concepts and roles, generality ordering between contexts.

3.2 Semantics

As exposed in the previous section, a $\mathcal{L}^{CT}$ consists of four classes of expressions: Ξ (context constructs), P and Γ (role and concept descriptions), $\mathcal{A}$ (assertions). Semantics of P and Γ will be the standard description logic semantics of roles and concepts, on which our framework is based. Semantics for Ξ will be given in terms of model theoretic operations on sets of description logic models, and at that stage the semantics of assertions $\mathcal{A}$ will be defined via an appropriate satisfaction relation. The structures obtained, which we call *contextual terminology models* or ct-models, provides a formal semantics for $\mathcal{L}^{CT}$ languages.

The firs step is then to provide the definition of a description logic model for a language $\mathcal{L}_i$ ([2]).

Definition 1. (Models for $\mathcal{L}_i$'s)
A model m for a language $\mathcal{L}_i$ is defined as follows:

$$m = \langle \Delta_m, \mathcal{I}_m \rangle$$

where:

– *Δ_m is the (non empty) domain of the model;*
– *$\mathcal{I}_m$ is a function $\mathcal{I}_m : \mathbf{A_i} \cup \mathbf{R_i} \longrightarrow \mathcal{P}(\Delta_m) \cup \mathcal{P}(\Delta_m \times \Delta_m)$, such that to every element of $\mathbf{A_i}$ and $\mathbf{R_i}$ an element of $\mathcal{P}(\Delta_m)$ and, respectively, of $\mathcal{P}(\Delta_m \times \Delta_m)$ is associated. This interpretation of atomic concepts and roles of $\mathcal{L}_i$ on Δ_m is then inductively extended:*

$$\begin{aligned}
\mathcal{I}_m(\top) &= \Delta_m \\
\mathcal{I}_m(\bot) &= \emptyset \\
\mathcal{I}_m(\neg\gamma) &= \Delta_m \setminus \mathcal{I}_m(\gamma) \\
\mathcal{I}_m(\gamma_1 \sqcap \gamma_2) &= \mathcal{I}_m(\gamma_1) \cap \mathcal{I}_m(\gamma_2) \\
\mathcal{I}_m(\forall \rho.\gamma) &= \{a \in \Delta_m \mid \forall b, < a, b > \in I_m(\rho) \Rightarrow b \in I_m(\gamma)\} \\
\mathcal{I}_m(\overline{\rho}) &= \Delta_m \times \Delta_m \setminus \mathcal{I}_m(\rho).
\end{aligned}$$

A model m for a language $\mathcal{L}_i$ assigns a denotation to each atomic concept (for instance the set of elements of Δ_m that instantiate the concept `bike`) and to each atomic role (for instance the set of pairs of Δ_m which are in a relation such that the first element is said to "drive" the second element of the pair). Accordingly, meaning is given to each complex concept (for instance the set of elements of Δ_m that instantiate the concept `vehicle` $\sqcup$ `bike`) and to each complex role (for instance the set of pairs listing elements related by role $\overline{\texttt{drive}}$).

3.3 Models for $\mathcal{L}^{CT}$

We can now define a notion of *contextual terminology model* (ct-model) for languages $\mathcal{L}^{CT}$.

Definition 2. (ct-models)
A ct-model $\mathbb{M}$ is a structure:

$$\mathbb{M} = \langle \{\mathbf{M_i}\}_{\mathbf{0 \leq i \leq n}}, \mathbb{I} \rangle$$

where:

– *$\{\mathbf{M}_i\}_{0 \leq i \leq n}$ is the family of the sets of models $\mathbf{M}_i$ of each language $\mathcal{L}_i$. That is, $\forall m \in \mathbf{M}_i$, m is a model for $\mathcal{L}_i$.*
– *$\mathbb{I}$ is a function $\mathbb{I} : \mathbf{c} \longrightarrow \mathcal{P}(\mathbf{M}_0) \cup \ldots \cup \mathcal{P}(\mathbf{M}_n)$. In other words, this function associates to each atomic context identifier in $\mathbf{c}$ a subset of the set of all models in some language $\mathcal{L}_i$: $\mathbb{I}(c) = M$ with $M \subseteq \mathbf{M}_i$ for some i s.t. $0 \leq i \leq n$. Function $\mathbb{I}$ can be seen as labeling sets of models on some language i via atomic context identifiers. Notice that $\mathbb{I}$ fixes, for each atomic context identifier, the language on which the context denoted by the identifier is specified. We could say that it is $\mathbb{I}$ itself which fixes a specific index for each atomic context identifier c.*
– *$\forall m', m'' \in \bigcup_{0 \leq i \leq n} \mathbf{M}_i$, $\Delta_{m'} = \Delta_{m''}$. That is, the domain of all models m is unique. We assume this constraint simply because we are interested in modeling different conceptualizations of a* same *set of individuals.*

Contexts are therefore formalized as sets of models for the same language. This perspective allows for straightforward model theoretical definitions of operations on contexts.

3.4 Context focus

We model focus as a specific operation on sets of models which provides the semantic counterpart for the *contextual focus* operator introduced in $\mathcal{L}^{CT}$. Intuitively, abstracting a context ξ to a language $\mathcal{L}_i$ yields a context consisting in that part of ξ which can be expressed in $\mathcal{L}_i$.

Let us first recall a notion of *domain restriction* ($\rceil$) of a function f w.r.t. a subset C of the domain of f. Intuitively, a domain restriction of a function f is nothing but the function $C\rceil f$ having C as domain and s.t. for each element of C, f and $C\rceil f$ return the same image: $C\rceil f = \{\langle x, f(x)\rangle \,|x \in C\}$.

Definition 3. (Context focus operation: $\rceil_i$)
Let M' be a set of models, then: $\rceil_i M' = \{m \mid m = \langle \Delta_{m'}, \mathbf{A}_i \cup \mathbf{R}_i \rceil \mathcal{I}_{m'}\rangle \text{ \& } m' \in M'\}$.

The following can be proved.

Proposition 1. (Properties of context focus)
Operation $\rceil_i$ is: surjective, idempotent ($\rceil_i(\rceil_i M) = \rceil_i M$), normal ($\rceil_i \emptyset = \emptyset$), additive ($\rceil_i(M_1 \cup M_2) = \rceil_i M_1 \cup \rceil_i M_2$), monotonic ($M_1 \subseteq M_2 \Rightarrow \rceil_i M_1 \subseteq \rceil_i M_2$).

Proof. A proof is worked out in [11].

The operation of focus allows for shifting from richer to simpler languages and it is, as we would intuitively expect: surjective (every context, even the empty one, can be seen as the result of focusing a different richer context, in the most trivial case, a focus of itself), idempotent (focusing on a focus yields the same first focus), normal (focusing the empty context yields the empty context), additive (the focus of a context obtained via joining of two contexts can be obtained also joining the focuses of the two contexts), monotonic (if a context is less general then another one, the focus of the first is also less general than the focus of the second one). Notice also that operation $\rceil_i$ yields the empty set of models when it is applied to a context M' the language of which is not an expansion of $\mathcal{L}_i$. This is indeed very intuitive: the context obtained via focus of the context "dinosaurs" on the language of, say, "gourmet cuisine" should be empty.

A detailed comparison of our account of focus with approaches available in the literature on context theory is discussed in [11].

3.5 Operations on contexts

We are now in a position to give a semantics to context constructs as introduced in Section 3.1. In Definition 2 atomic contexts are interpreted as sets of models on some language $\mathcal{L}_i$ for $0 \leq i \leq n$: $\mathbb{I}(c) = M \in \mathcal{P}(\mathbf{M}_0) \cup \ldots \cup \mathcal{P}(\mathbf{M}_n)$. The semantics of context constructs Ξ can be defined via inductive extension of that definition.

Definition 4. (Semantics of context constructs)
Let ξ, ξ_1, ξ_2 be context constructs, then:

$$\begin{aligned}
&\mathbb{I}(\mathrm{fcs}_i\ \xi) = \rceil_i \mathbb{I}(\xi)\\
&\mathbb{I}(\bot_i) = \emptyset\\
&\mathbb{I}(\top_i) = \mathbf{M}_i\\
&\mathbb{I}(\xi_1 \curlyvee_i \xi_2) = \rceil_i(\mathbb{I}(\xi_1)\ \cup\ \mathbb{I}(\xi_2)).
\end{aligned}$$

The focus operator fcs_i is interpreted on the contextual focus operation introduced in Definition 3, i.e., as the restriction of the interpretation of its argument to language $\mathcal{L}_i$. The $\bot_i$ context is interpreted as the empty context (the same on each language); the $\top_i$ context is interpreted as the greatest, or most general, context on $\mathcal{L}_i$; the binary $\curlyvee_i$-composition of contexts is interpreted as the lowest upper bound of the restriction of the interpretations of the two contexts on $\mathcal{L}_i$.

In [15] the statement about the need for addressing "contexts as abstract mathematical entities" was set forth. Here, moving from an analysis of contextual terminologies, we develop an account of context interplay based on model theoretic operations. In some sense, we propose a view on contexts as "algebraic entities". In fact, it is easy to prove ([11]) that contexts, as conceived here, are structured according to a Boolean Algebra with Operators ([14]). This observation distills the type of conception of context we hold here: contexts are sets of models on different concept description languages; on each language the set of possible contexts is structured in a Boolean Algebra; adding operations of focus on a finite number of sublanguages yields a Boolean Algebra with Operators.

3.6 Assertions

The semantics of assertions is defined as follows.

Definition 5. (Semantics of assertions: $\models$)
Let ξ, ξ_1, ξ_2 be a context constructs, $\gamma, \gamma_1, \gamma_2$ concept description, then:

$$\begin{aligned}
&\mathbb{M} \models \gamma \downarrow \xi \quad \textit{iff} \quad \{\mathrm{D}_c \mid \langle \gamma, \mathrm{D}_c \rangle \in \mathcal{I}_m \ \&\ m \in \mathbb{I}(\xi)\} \neq \emptyset && (1)\\
&\mathbb{M} \models \rho \downarrow \xi \quad \textit{iff} \quad \{\mathrm{D}_r \mid \langle \rho, \mathrm{D}_r \rangle \in \mathcal{I}_m \ \&\ m \in \mathbb{I}(\xi)\} \neq \emptyset && (2)\\
&\mathbb{M} \models \xi : \gamma_1 \sqsubseteq \gamma_2 \quad \textit{iff} \quad \mathbb{M} \models \gamma_1 \downarrow \xi, \mathbb{M} \models \gamma_2 \downarrow \xi\\
&\qquad\qquad\qquad\qquad \textit{and}\ \forall m \in \mathbb{I}(\xi)\ \mathcal{I}_m(\gamma_1) \subseteq \mathcal{I}_m(\gamma_2) && (3)\\
&\mathbb{M} \models \xi : \rho_1 \sqsubseteq \rho_2 \quad \textit{iff} \quad \mathbb{M} \models \rho_1 \downarrow \xi, \mathbb{M} \models \rho_2 \downarrow \xi\\
&\qquad\qquad\qquad\qquad \textit{and}\ \forall m \in \mathbb{I}(\xi)\ \mathcal{I}_m(\rho_1) \subseteq \mathcal{I}_m(\rho_2) && (4)\\
&\mathbb{M} \models \xi_1 \preccurlyeq \xi_2 \quad \textit{iff} \quad \mathbb{I}(\xi_1) \subseteq \mathbb{I}(\xi_2). && (5)
\end{aligned}$$

Clauses (1) and (2) specify when a concept, respectively a role, is meaningful with respect to a context. This is the case when the set of denotations D_c and D_r which the models constituting the context attribute to that concept (D_c being a set of elements of the domain) or that role (D_r being a set of pairs of elements of the domain), is not empty. If concept γ is not expressible in the language of context ξ, then concept γ gets no denotation at all in context ξ. This happens simply because concept γ does not belong to the domain of functions $\mathcal{I}_m$, and there therefore exists no interpretation for that concept in the models constituting ξ. The same holds for a role ρ. Clauses (3) and (4) deal with satisfaction of contextual subsumptions. A contextual concept subsumption relation between γ_1 and γ_2 holds iff concepts γ_1 and γ_2 are defined in the models constituting context ξ, i.e., they receive a denotation in those models, and all the description logic models constituting that context interpret γ_1 as a subconcept of γ_2. Note that this is precisely the clause for the validity of a subsumption relation in standard description logics, but together with the fact that the concepts involved are actually meaningful in that context. Intuitively, we interpret contextual subsumption relations as inherently presupposing the meaningfulness of their terms[10]. A perfectly analogous observation holds also for the clause regarding contextual role subsumption relations. Clause (5) gives a semantics to the $\preccurlyeq$ relation between context constructs interpreting it as a standard subset relation: $\xi_1 \preccurlyeq \xi_2$ means that context denoted by ξ_1 contains at most all the models that ξ_2 contains, that is to say, ξ_1 is *at most as general as* ξ_2. Clauses for boolean connectives are the obvious ones.

Notions of validity and logical consequence are classically defined. An assertion α is valid if all ct-models $\mathbb{M}$ satisfy it. An assertion α is a logical consequence of the set of assertions $\alpha_1, \ldots, \alpha_n$ if all ct-models satisfying $\alpha_1, \ldots, \alpha_n$ satisfy α.

4 Contextual Terminologies at Work

4.1 Formalizing the first scenario

We are now in the position to formalize Example 1.

Example 3. **(Sufficient conditions for "liability")** To formalize the first scenario within our setting a language $\mathcal{L}$ is needed, which contains the following atomic concepts: `person`, `liable`, `vehicle`, `car`, `bicycle`; and the following atomic roles: drive and wheel. Four atomic contexts are at issue here: the context of the main regulation R, let us call it c_R; the contexts of the municipal regulations M1, M2 and M3, let us call them c_{M1}, c_{M2} and c_{M3} respectively. These contexts should be interpreted on two relevant languages (let us call them $\mathcal{L}_0$ and $\mathcal{L}_1$) s.t. $\mathbf{A}_0 = \{\texttt{person}, \texttt{liable}, \texttt{vehicle}\}$, $\mathbf{R}_0 = \{\text{drive}\}$ and $\mathbf{A}_1 = \{\texttt{person}, \texttt{liable}, \texttt{vehicle}, \texttt{car}, \texttt{bicycle}\}$, $\mathbf{R}_1 = \{\text{drive}, \text{wheel}\}$. That is to say, an abstract language concerning only persons, liability, vehicles and the action of driving, and a more detailed language concerning, besides liable persons, vehicles and driving, also cars, bicycles and the action of wheeling. The sets of all models for $\mathcal{L}_0$ and $\mathcal{L}_1$ are then respectively $\mathbf{M}_0$ and $\mathbf{M}_1$.

[10] For a more detailed discussion of these clauses we refer the reader to [12].

To model the desired situation, our ct-model should then at least satisfy the following $\mathcal{L}^{CT}$ formulas:

$$c_{M1} \curlyvee_0 c_{M2} \curlyvee_0 c_{M3} \preccurlyeq c_R \tag{6}$$
$$\sim (\texttt{car} \downarrow c_R) \wedge \sim (\texttt{bicycle} \downarrow c_R) \wedge \sim (\text{wheel} \downarrow c_R) \tag{7}$$
$$c_R : \texttt{person} \sqcap \exists \text{drive}.\texttt{vehicle} \sqsubseteq \texttt{person} \sqcap \texttt{liable} \tag{8}$$
$$c_{M1} \curlyvee_1 c_{M2} \curlyvee_1 c_{M3} : \texttt{car} \sqsubset \texttt{vehicle} \tag{9}$$
$$c_{M1} \curlyvee_1 c_{M2} : \texttt{bicycle} \sqsubset \texttt{vehicle} \tag{10}$$
$$c_{M3} : \texttt{bicycle} \sqsubseteq \neg\texttt{vehicle} \tag{11}$$
$$c_{M1} \curlyvee_1 c_{M3} : \text{wheel} \sqsubset \text{drive} \tag{12}$$
$$c_{M2} : \text{wheel} \sqsubseteq \overline{\text{drive}} \tag{13}$$
$$c_{M1} \curlyvee_1 c_{M2} \curlyvee_1 c_{M3} : \exists \text{wheel}.\texttt{car} \equiv \bot \tag{14}$$

Formula (6) plays a key role, stating that the three contexts c_{M1}, c_{M2}, c_{M3} are concrete variants of context c_R. It tells this by saying that the context obtained by joining the three concrete contexts on language $\mathcal{L}_0$ (the language of c_R) is at most as general as context c_R, that is: $\rceil_0\mathbb{I}(c_{M1}) \cup \rceil_0\mathbb{I}(c_{M2}) \cup \rceil_0\mathbb{I}(c_{M3}) \subseteq \mathbb{I}(c_R)$ (see Section 3.2). As we will see in the following, this makes c_{M1}, c_{M2} and c_{M3} inherit what holds in c_R. Formula (7) specifies what concepts and roles do not get interpretation in the abstract context c_R. Formula (8) formalizes the abstract rule to the effect that persons driving vehicles (within public parks) are liable for a violation of the applicable regulation. Formulas (9)-(11) describe the different taxonomies holding in the three concrete contexts at issue, while formulas (12) and (13) describe the different role hierarchies holding in those contexts. The last formula can be seen as simply stating some background knowledge to the effect that to wheel a car is an empty concept.

To discuss in some more depth the proposed formalization, let us first list some interesting logical consequences of formulas (6)-(14). We will focus on subsumptions contextualized to monadic contexts, that is to say, we will show what the consequences of formulas (6)-(14) are at the level of the three contexts c_{M1}, c_{M2} and c_{M3} considered in isolation.

$$\begin{aligned}
(6,8) &\models c_{M1} : \texttt{person} \sqcap \exists \text{drive}.\texttt{vehicle} \sqsubseteq \texttt{person} \sqcap \texttt{liable} \\
(9) &\models c_{M1} : \texttt{person} \sqcap \exists \text{drive}.\texttt{car} \sqsubset \texttt{person} \sqcap \exists \text{drive}.\texttt{vehicle} \\
(10) &\models c_{M1} : \texttt{person} \sqcap \exists \text{drive}.\texttt{bicycle} \sqsubset \texttt{person} \sqcap \exists \text{drive}.\texttt{vehicle} \\
(10,12) &\models c_{M1} : \texttt{person} \sqcap \exists \text{wheel}.\texttt{bicycle} \sqsubset \texttt{person} \sqcap \exists \text{drive}.\texttt{bicycle} \\
(6,8,10) &\models c_{M1} : \texttt{person} \sqcap \exists \text{drive}.\texttt{bicycle} \sqsubset \texttt{person} \sqcap \texttt{liable} \\
(6,8,10,12) &\models c_{M1} : \texttt{person} \sqcap \exists \text{wheel}.\texttt{bicycle} \sqsubset \texttt{person} \sqcap \texttt{liable}
\end{aligned}$$

$$\begin{aligned}
(6,8) &\vDash c_{M2} : \mathtt{person} \sqcap \exists \mathrm{drive}.\mathtt{vehicle} \sqsubseteq \mathtt{person} \sqcap \mathtt{liable}\\
(9) &\vDash c_{M2} : \mathtt{person} \sqcap \exists \mathrm{drive}.\mathtt{car} \sqsubseteq \mathtt{person} \sqcap \exists \mathrm{drive}.\mathtt{vehicle}\\
(10) &\vDash c_{M2} : \mathtt{person} \sqcap \exists \mathrm{drive}.\mathtt{bicycle} \sqsubset \mathtt{person} \sqcap \exists \mathrm{drive}.\mathtt{vehicle}\\
(13) &\vDash c_{M2} : \mathtt{person} \sqcap \exists \mathrm{wheel}.\mathtt{bicycle} \sqsubseteq \mathtt{person} \sqcap \exists \overline{\mathrm{drive}}.\mathtt{bicycle}\\
(10,13) &\vDash c_{M2} : \mathtt{person} \sqcap \exists \mathrm{drive}.\mathtt{bicycle} \sqsubseteq \mathtt{person} \sqcap \exists \overline{\mathrm{drive}}.\mathtt{vehicle}\\
(6,8,10) &\vDash c_{M2} : \mathtt{person} \sqcap \exists \mathrm{drive}.\mathtt{bicycle} \sqsubset \mathtt{person} \sqcap \mathtt{liable}\\
(6,8) &\vDash c_{M2} : \mathtt{person} \sqcap \neg\mathtt{liable} \sqsubseteq \mathtt{person} \sqcap \neg\exists \mathrm{drive}.\mathtt{vehicle}\\
(6,8,9,10,13) &\vDash \sim c_{M2} : \mathtt{person} \sqcap \exists \mathrm{wheel}.\mathtt{bicycle} \sqsubset \mathtt{person} \sqcap \mathtt{liable}
\end{aligned}$$

$$\begin{aligned}
(6,8) &\vDash c_{M3} : \mathtt{person} \sqcap \exists \mathrm{drive}.\mathtt{vehicle} \sqsubseteq \mathtt{person} \sqcap \mathtt{liable}\\
(9) &\vDash c_{M3} : \mathtt{person} \sqcap \exists \mathrm{drive}.\mathtt{car} \sqsubseteq \mathtt{person} \sqcap \exists \mathrm{drive}.\mathtt{vehicle}\\
(11) &\vDash c_{M2} : \mathtt{person} \sqcap \exists \mathrm{drive}.\mathtt{bicycle} \sqsubset \mathtt{person} \sqcap \exists \mathrm{drive}.\neg\mathtt{vehicle}\\
(12) &\vDash c_{M3} : \mathtt{person} \sqcap \exists \mathrm{wheel}.\mathtt{bicycle} \sqsubset \mathtt{person} \sqcap \exists \mathrm{drive}.\mathtt{bicycle}\\
(6,8) &\vDash c_{M3} : \mathtt{person} \sqcap \neg\mathtt{liable} \sqsubseteq \mathtt{person} \sqcap \neg\exists \mathrm{drive}.\mathtt{vehicle}\\
(6,8,11) &\vDash \sim c_{M3} : \mathtt{person} \sqcap \exists \mathrm{drive}.\mathtt{bicycle} \sqsubset \mathtt{person} \sqcap \mathtt{liable}\\
(6,8,11,12) &\vDash \sim c_{M3} : \mathtt{person} \sqcap \exists \mathrm{wheel}.\mathtt{bicycle} \sqsubset \mathtt{person} \sqcap \mathtt{liable}
\end{aligned}$$

These are indeed the formulas that we would intuitively expect to hold in our scenario. The list displays three sets of formulas grouped on the basis of the context to which they pertain. Let us have a closer look to them. The first consequence of each group results from the generality relation expressed in (6), by means of which, the content of (8) is shown to hold also in the three concrete contexts: in simple words, contexts c_{M1}, c_{M2} and c_{M3} inherit the general rule stating the liability of persons driving vehicles (within public parks). Via this inherited rule, and via (9), it is shown that, in all contexts, who drives a car is also held liable (second consequence of each group). As to cars and driving cars then, all contexts agree. Where differences arise is in relation with how the concept of bicycle and the role of wheeling are handled.

In context c_{M1}, we have that it does not matter if somebody wheels or actually drives a bicycle, because in both cases this would count as driving a vehicle, and therefore of violating the regulation. In fact, in this context, a bicycle is a vehicle (10) and to wheel is a way of driving (11). Context c_{M2}, instead, expresses a different view. Since bicycles count as vehicles (10), to drive a bicycle is still a ground for liability. On the other hand, to wheel is actually classified as a way of refraining from driving (13), and therefore, persons wheeling bicycles do not count as persons driving vehicles, and do not commit any violation. Context c_{M3} yields yet another terminology. Here bicycles are classified as objects which are not vehicles (11). Therefore, although to wheel is conceived as a way of driving (11), both to drive and to wheel a bicycle does not determine liability. With respect to this, it is instructive to notice that even though both in c_{M2} and c_{M3} to wheel a bicycle is not a sufficient reason for being held liable, this holds for two different reasons: in c_{M2} because of (13), and in c_{M3} because of

(11). This illustrates how our framework is able to cope with some quite subtle nuances that characterize contextual classifications.

4.2 A model of the scenario

In this section we expose a simple ct-model satisfying (6)-(14). Let us stipulate that the models m that constitute our interpretation of contexts identifiers consist of a domain $\Delta_m = \{a, b, c, d, e, f, g\}$. Being $\mathcal{L}_0$ and $\mathcal{L}_1$ the two languages at issue, the domain of the ct-models is $\mathbf{M}_0 \cup \mathbf{M}_1$. A ct-model would then be, for instance, a structure $\langle \mathbf{M}_0 \cup \mathbf{M}_1, \mathbb{I} \rangle$ where $\mathbb{I}$ is such that:

- $\mathbb{I}(c_{M1}) = \{m_1, m_2\} \subseteq \mathbf{M}_1$ s.t. $\mathcal{I}_{m_1}(\texttt{person}) = \{e, f, g\}$, $\mathcal{I}_{m_1}(\texttt{vehicle}) = \{a, b, c, d\}$, $\mathcal{I}_{m_1}(\texttt{bicycle}) = \{a, b\}$, $\mathcal{I}_{m_1}(\texttt{car}) = \{c, d\}$, $\mathcal{I}_{m_1}(\text{drive}) = \{< e, a >, < f, c >\}$, $\mathcal{I}_{m_1}(\text{wheel}) = \{< e, a >\}$, $\mathcal{I}_{m_1}(\texttt{liable}) = \{e, f\}$ and $\mathcal{I}_{m_2}$ agrees with $\mathcal{I}_{m_1}$ on the interpretation of `person`, `bicycle`, `car`, `vehicle` and $\mathcal{I}_{m_2}(\text{drive}) = \{< f, c >, < g, d >\}$, $\mathcal{I}_{m_2}(\text{wheel}) = \{< g, d >\}$, $\mathcal{I}_{m_2}(\texttt{liable}) = \{f, g\}$.
- $\mathbb{I}(c_{M2}) = \{m_3, m_4\} \subseteq \mathbf{M}_1$ s.t. $\mathcal{I}_{m_3}$ and $\mathcal{I}_{m_4}$ agree with $\mathcal{I}_{m_1}$ on the interpretation of `person`, `bicycle`, `car`, `vehicle` and $\mathcal{I}_{m_3}(\text{drive}) = \{< f, d >, < g, a >\}$, $\mathcal{I}_{m_3}(\text{wheel}) = \{< e, a >\}$, $\mathcal{I}_{m_3}(\texttt{liable}) = \{f, g\}$ and $\mathcal{I}_{m_4}(\text{drive}) = \{< e, c >\}$, $\mathcal{I}_{m_4}(\text{wheel}) = \{< f, a >\}$, $\mathcal{I}_{m_4}(\texttt{liable}) = \{e\}$.
- $\mathbb{I}(c_{M3}) = \{m_5\} \subseteq \mathbf{M}_1$ s.t. $\mathcal{I}_{m_5}$ agrees with $\mathcal{I}_{m_1}$ on the interpretation of `person`, `bicycle`, `car` and $\mathcal{I}_{m_5}(\texttt{vehicle}) = \{c, d\}$, $\mathcal{I}_{m_5}(\text{drive}) = \{< e, a >, < f, c >, < g, d >\}$, $\mathcal{I}_{m_1}(\text{wheel}) = \{< e, a >\}$, $\mathcal{I}_{m_1}(\texttt{liable}) = \{f, g\}$.
- $\mathbb{I}(c_R) = \{m \mid m = \langle \Delta_m, \mathbf{A}_0 \cup \mathbf{R}_0 \rceil \mathcal{I}_i \rangle$ and $1 \leq i \leq 5\}$, that is, c_R is interpreted by the model as the union of all models constituting c_{M1}, c_{M2} and c_{M3} restricted to the language $\mathcal{L}_0$.

The model makes an interesting feature of our semantics explicit. In contexts c_{M1} and c_{M2} the set of liable persons do not coincide in the two models constituting the context; nevertheless only persons driving vehicles are indeed liable. This clearly shows that contexts can be viewed as clusters of possible situations all instantiating the same terminology[11].

4.3 Formalizing the second scenario

The formalization of the scenario introduced in Example 2 follows.

Example 4. **(Categorizing "teenagers on skates")** The global language $\mathcal{L}$ needed contains the following atomic concepts: `conv`, `person`, `obj`, `vehicle`, `teenager`, `skate`; and the following atomic role: transp. Three are the atomic contexts at issue here: the context of the main regulation R, let us call it c_R; the contexts of the municipal regulations M1 and M2, let us call them c_{M1} and c_{M2} respectively. These contexts should be interpreted on two relevant languages (let

[11] We developed this intuition also in a modal logic setting modeling contexts as sets of possible worlds. See [13].

us call them $\mathcal{L}_0$ and $\mathcal{L}_1$) s.t. $\mathbf{A}_0 = \{\texttt{conv}, \texttt{person}, \texttt{obj}, \texttt{vehicle}\}$, $\mathbf{R}_0 = \{\text{transp}\}$ and $\mathbf{A}_1 = \mathbf{A}_0 \cup \{\texttt{self\_prop}, \texttt{teenager}, \texttt{skate}\}$, $\mathbf{R}_1 = \mathbf{R}_0$. That is to say, an abstract language concerning only conveyances, persons, objects, vehicles and the attribute of transporting, and a more detailed language concerning, besides this, also teenagers and skates. The sets of all models for $\mathcal{L}_0$ and $\mathcal{L}_1$ are then respectively $\mathbf{M}_0$ and $\mathbf{M}_1$. To model the desired situation, a ct-model should then at least satisfy the following $\mathcal{L}^{CT}$ formulas:

$$c_{M1} \curlyvee_0 c_{M2} \preccurlyeq c_R \tag{15}$$
$$\sim (\texttt{teenager} \downarrow c_R) \wedge \sim (\texttt{skate} \downarrow c_R) \tag{16}$$
$$c_R : \texttt{vehicle} \sqsubseteq \texttt{conv} \sqcap \forall\text{transp}.(\texttt{person} \sqcup \texttt{obj}) \tag{17}$$
$$c_{M1} : \texttt{vehicle} \sqsubseteq \texttt{self\_prop} \tag{18}$$
$$c_{M1} : \texttt{conv} \sqcap \forall\text{transp}.(\texttt{person} \sqcup \texttt{obj}) \sqcap \texttt{self\_prop} \sqsubseteq \texttt{vehicle} \tag{19}$$
$$c_{M2} : \texttt{conv} \sqcap \forall\text{transp}.(\texttt{person} \sqcup \texttt{obj}) \sqsubseteq \texttt{vehicle} \tag{20}$$
$$c_{M1} \curlyvee_1 c_{M2} : \texttt{teenager} \sqsubset \texttt{person} \tag{21}$$
$$c_{M1} \curlyvee_1 c_{M2} : \texttt{skate} \sqsubset \texttt{conv} \tag{22}$$
$$c_{M1} \curlyvee_1 c_{M2} : \texttt{skate} \sqsubset \neg\texttt{self\_prop} \tag{23}$$

For reason of space, we cannot discuss this example in as many details as the previous one. We stress the most important aspects. Formulas (15) and (16) are the analogous of formulas (6) and (7). Formula (17) represents the abstract constraints that context c_R imposes on the concept `vehicle`.

Formulas (18), (19) and (20) express the additional constraints on the concept `vehicle` holding in context c_{M1} and c_{M2} respectively: both contexts specify sufficient conditions and context c_{M1} adds also new necessary ones (18). Formulas (21) and (22) state the intuitive background knowledge common to the two concrete contexts. The point of the scenario consists in showing how teenagers on skateboards are conceptualized in the three contexts, that is to say: how are concept $\texttt{skate} \sqcap \exists\text{transp}.\texttt{teenager}$ and concept `vehicle` related in each context? This can be easily shown via some relevant logical consequences of (15)-(23):

$$(15, 17, 18, 19) \models c_{M1} : \texttt{conv} \sqcap \forall\text{transp}.(\texttt{person} \sqcup \texttt{obj}) \sqcap \texttt{self\_prop} \equiv \texttt{vehicle}$$
$$(15, 17, 18, 19, 21, 22, 23) \models c_{M1} : \texttt{skate} \sqcap \exists\text{transp}.\texttt{teenager} \sqsubseteq \neg\texttt{vehicle}$$
$$(15, 17, 20) \models c_{M2} : \texttt{conv} \sqcap \forall\text{transp}.(\texttt{person} \sqcup \texttt{obj}) \equiv \texttt{vehicle}$$
$$(15, 17, 20, 21, 22) \models c_{M2} : \texttt{skate} \sqcap \exists\text{transp}.\texttt{teenager} \sqsubseteq \texttt{vehicle}.$$

These formulas show that in the two concrete contexts two different definitions of `vehicle` hold, and therefore two different conceptualizations of concepts such as $\texttt{skate} \sqcap \exists\text{transp}.\texttt{teenager}$: since skateboards are, in c_{M1}, non self-propelled, even if they are conveyances transporting people, they are not classifiable as vehicles .

5 Conclusions

We motivated and devised a formal framework for representing contextual ontologies via a contextualized version of description logic semantics. The key idea has been to show that the basic intuition of understanding contexts as sets of description logic models, which we presented in [12], works smoothly also with subsumption statements of more complex concept descriptions. The next step will be to side contextual terminologies with appropriate contextual assertion boxes (ABoxes) in which to reason about contextual instantiations of concepts and roles.

In future work we will focus on developing a proof theory along the lines sketched in [11] and on analyzing the complexity of the framework especially with respect to inter-contextual reasoning. We intend also to apply contextual terminologies to the study of the contextual meaning of actions in agents institutions: raising a hand during a bid has a different meaning than raising a hand during a scientific workshop. A natural way to do this, would be to exploit established results about the relation between dynamic logic and description logic ([4]) to get to a contextualized form of dynamic logic.

Acknowledgments

We would like to thank the anonymous reviewers of CLIMA VI for their valuable comments.

References

1. V. Akman and M. Surav. Steps toward formalizing context. *AI Magazine*, 17(3):55–72, 1996.
2. F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider. *The Description Logic Handbook.* Cambridge University Press, Cambridge, 2002.
3. M. Benerecetti, P. Bouquet, and C. Ghidini. Contextual reasoning distilled. *Journal of Experimental and Theoretical Artificial Intelligence (JETAI)*, 12(3):279–305, 2000.
4. G. De Giacomo and M. Lenzerini. TBox and ABox reasoning in expressive description logics. In L. Carlucci Aiello, J. Doyle, and S. Shapiro, editors, *KR'96: Principles of Knowledge Representation and Reasoning*, pages 316–327. Morgan Kaufmann, San Francisco, California, 1996.
5. F. Dignum. Agents, markets, institutions, and protocols. In *Agent Mediated Electronic Commerce, The European AgentLink Perspective.*, pages 98–114. Springer-Verlag, 2001.
6. F. Dignum. Abstract norms and electronic institutions. In *Proceedings of the International Workshop on Regulated Agent-Based Social Systems: Theories and Applications (RASTA '02), Bologna*, pages 93–104, 2002.
7. C. Ghidini and F. Giunchiglia. Local models semantics, or contextual reasoning = locality + compatibility. *Artificial Intelligence*, 127(2):221–259, 2001.

8. C. Ghidini and F. Giunchiglia. A semantics for abstraction. In R.López de Mántaras and L. Saitta, editors, *Proceedings of ECAI'2004, including PAIS 2004*, pages 343–347, 2004.
9. D. Grossi, H. Aldewereld, J. Vázquez-Salceda, and F. Dignum. Ontological aspects of the implementation of norms in agent-based electronic institutions. In *Proceedings of NorMAS'05, Symposium on normative multi-agent systems.*, Hatfield, England, april 2005.
10. D. Grossi and F. Dignum. From abstract to concrete norms in agent institutions. In M. G. et al. Hinchey, editor, *Formal Approaches to Agent-Based Systems: Third International Workshop, FAABS 2004*, Lecture Notes in Computer Science, pages 12–29. Springer-Verlag, April 2004.
11. D. Grossi, F. Dignum, and J-J. Ch. Meyer. Context in categorization. In *Proceedings of CRR'05, International Workshop on Context Representation and Reasoning*, Paris, July 2005.
12. D. Grossi, F. Dignum, and J-J. Ch. Meyer. Contextual taxonomies. In J. Leite and P. Toroni, editors, *Proceedings of CLIMA V Workshop, Lisbon, September*, LNAI 3487, pages 33–51. Springer, 2005.
13. D. Grossi, J-J. Ch. Meyer, and F. Dignum. Modal logic investigations in the semantics of counts-as. To be presented at ICAIL'05, Tenth International Conference on Artificial Intelligence and Law, Bologna, June 2005.
14. B. Jónsson and A. Tarski. Boolean algebras with operators: Part I. *American Journal of Mathematics*, 73:891–939, 1951.
15. J. McCarthy. Notes on formalizing contexts. In T. Kehler and S. Rosenschein, editors, *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 555–560, Los Altos, California, 1986. Morgan Kaufmann.
16. J. Searle. *The Construction of Social Reality.* Free Press, 1995.
17. Y. Shoham. Varieties of context. pages 393–407. Academic Press Professional, Inc., 1991.
18. J. Vázquez-Salceda. *The role of Norms and Electronic Institutions in Multi-Agent Systems.* Birkhuser Verlag AG, 2004.
19. J. Vázquez-Salceda and F. Dignum. Modelling electronic organizations. In J. Muller V. Marik and M. Pechoucek, editors, *Proceedings CEEMAS'03. LNAI 2691*, pages 584–593, Berlin, 2003. Springer-Verlag.

Contract-related Agents

John Knottenbelt and Keith Clark

Dept of Computing, Imperial College London,
180 Queens Gate, London, SW7 2AZ, UK
{jak97,klc}@imperial.ac.uk

Abstract. We propose a simple event calculus representation of contracts and a reactive belief-desire-intention agent architecture to enable the monitoring and execution of contract terms and conditions. We use the event calculus to deduce current and past obligations, obligation fulfilment and violation. By associating meta-information with the contracts, the agent is able to select which of its contracts with other agents are relevant to solving its goals by outsourcing. The agent is able to handle an extendable set of contract types such as standing contracts, purchase contracts and service contracts without the need for a first-principles planner.

1 Introduction

Multi-agent systems is a growing research area and has already started to find application in industry in web services and the semantic web. There is also increased interest in agent coordination and choreography. Our approach sees contracts as a means of formally describing the relationships between agents in terms of obligations and permissions, as well as providing a coordination function.

By expressing the terms and conditions of a contract as a set of event-based rules – and so long as the participating agents agree on the history of events relevant to their contracts – an agent is able to obtain a completely unambiguous and indisputable view of the state of the contract at any given point in time.

We claim that the AgentSpeak(L)[11] architecture, with relatively few extensions, enables an agent to behave in a reactive manner (as is the case with service agents, where they react to obligations imposed on them) or a proactive manner where it makes use of agreed or newly proposed client contracts in order to impose obligations on other agents to do things for it. It may do this both to satisfy its own goals, or to discharged obligations it has arising from other contracts.

Starting off with a description of how contracts may be represented in the event calculus, we give an example of a short-term contract to conduct a purchase and a long-term standing contract to set-up short-term purchase contracts. In section 3 we discuss how the agents may communicate with each other and in section 4 we show how these communications can be used to effect the contract state (such as established facts, obligation fulfilment and violation). Section 5 briefly presents the AgentSpeak(L) architecture and how we have used and extended it to incorporate reasoning about contracts. We give the plan libraries for the customer and vendor agents which are able to monitor a general class of purchase and standing contracts, of which the contracts presented in section 2 are instances. Finally we review related work and concluding remarks.

2 Contract Representation

The core of the contract representation language is the event calculus [10], where communications are events and the contract rules specify how the events initiate and terminate obligation fluents. We make an implicit assumption that an agent is *permitted* to perform any communication that, taking into account the history of the use of the contract up to this point, will initiate an obligation on another party to the contract. This assumption has been sufficient for the examples studied so far, however, we do intend to investigate explicit representation of permissions in future work. In this paper we are using the Prolog variable syntax convention where variables begin with an uppercase letter.

We are using the full event-calculus[12] without the `releases` predicate since the examples we have considered so far do not require the use of non-inertial fluents. We also dispense with the `initiallyP` and `initiallyN` predicates by providing a contract start event, and writing `initiates(start, F, T)` and `terminates (start, F, T)` respectively. Figure 1 summarises the axioms of the event calculus.

```
holdsAt(F,T) ↔ happens(E,T1) ∧ T1<T ∧
  initiates(E,F,T1) ∧ not clipped(T1,F,T).
notHoldsAt(F,T) ↔ happens(F, T1) ∧ T1<T ∧
  terminates(E, F, T1) ∧ not declipped(T1,F,T).
clipped(T0,F,T1) ↔ ∃T[T0≤T ∧ T<T1 ∧ terminates(E,F,T)].
declipped(T0,F,T1) ↔ ∃T[T0≤T ∧ T<T1 ∧ initiates(E,F,T)].
```

Fig. 1. Event Calculus Summary

The body of a contract is represented by a binary relation, `contractClause`, between the label of the contract and the clauses belonging to the contract. Variables can be shared between the contract label and the clauses – those appearing in the label are conceptually parameters to the contract. If we were representing the contracts directly in Prolog, there would be one `contractClause` definition for each rule of the contract. For example, Figure 2 shows the first rule of a short term contract about a purchase transaction.

```
contractClause(
  customerVendorContract_purchase(customer[1]:C, vendor:V |
     vendorBank:VB, customerBank:CB, deliveryService:DS, item:I, price:P),
  initiates(start, oblig(V, achieve(value(invoice-no, _)), T+100), T).
```

Fig. 2. Contract Clause as Prolog

[1] To aid readability we employ the syntax `field:Value` to indicate a named field or parameter.

The label of the contract is `customerVendorContract_purchase(...)`. The parameters to the left of the `|` are the principals of the contract, usually the offeror and the offeree in a normal two party contract. The rule reads that at the start of the contract, the vendor (`V`) is obliged to announce the invoice number (`invoice-no`) relating to the purchase within 100 time units. `start` is the event marking the start of the contract's lifetime. `oblig` is a 3-place fluent relation between the bearer, the goal to be performed and its deadline. `achieve` indicates a state of affairs is to be achieved. `value` is a binary fluent relating contract variables to their values.

For ease of presentation, we adopt a more compact syntax, where the label is written once at the beginning of the contract, and the rules are written inside the following brace delimited block. Macro definitions for frequently used terms are written in small caps and marked with ≡ and should be textually substituted by the reader as they occur.

2.1 Short-term Contracts

Figure 3 shows the full text for a purchase contract, parameterised by the item being purchased, the price, the vendor, the customer, the vendor's and customer's bank, and a delivery service.

```
customerVendorContract_purchase(
     customer:C, vendor:V |
     vendorBank:VB, customerBank:CB, deliveryService:DS, item:I, price:P) {
  PAID(R) ≡ paid(payer:C, payee:V, price:P, reference:R).
  DELIVERED(R) ≡ delivered(item:I, destination:C, invoice-no:R).
  INVOICEOBLIG(DL) ≡ oblig(V, achieve(value(invoice-no, _), DL).
  PAYOBLIG(R, DL) ≡ oblig(C, achieve(PAID(R)), DL).
  DELIVEROBLIG(R, DL) ≡ oblig(V, achieve(DELIVERED(R)), DL).

  initiates(start, INVOICEOBLIG(T+100), T).
  initiates(E, PAYOBLIG(R, T+100), T) ← initiates(E, value(invoice-no, R), T).
  initiates(E, DELIVEROBLIG(R, T+300), T) ← initiates(E, value(invoice-no, R), T).
  initiates(E, owns(owner:C, item:I), T) ←
    holdsAt(value(invoice-no, R), T) ∧ initiates(E, fulfilled(PAYOBLIG(R, _))).

  terminates(E, owns(owner:V, item:I), T) ←
    holdsAt(value(invoice-no, R), T) ∧ initiates(E, fulfilled(PAYOBLIG(R, _))).

  authoritative(V, value(invoice-no, _)),
  authoritative(VB, PAID(_)).
  authoritative(CB, PAID(_)).
  authoritative(DS, DELIVERED(_)).
}
```

Fig. 3. Simple Purchase Contract

The first `initiates` rule, as described above, obliges the vendor to determine an invoice number and to signal this as an event notification for the contract. The vendor does this by sending an inform message to the customer (see sections 3 and 4.1). When the invoice number has been notified, the second and third `initiates` rules oblige the customer to concurrently pay within 100 time units and the vendor to deliver within 300 time units and to signal completion as contract related events.

The last `initiates` and the only `terminates` rule indicate that the customer will become the new owner of the item when the payment obligation has been fulfilled.

The `authoritative` clauses indicate which agents are authoritative for which fluents. The concept is related to controllable propositions[7] — the difference being that in our work the controlled proposition is limited in scope to a specific contract, rather than to the entire agent society. In the example, only the vendor has the authority to notify an invoice number. A notification by the customer would not be considered a contract event.

This mechanism is also useful to identify trusted-third parties: the banks are authoritative for the paid fluents (that is any payment from customer to vendor) and the delivery agent is authoritative for the proof of delivery fluent. An agent that is not authoritative for a fluent may attempt to communicate it, but the communication would have no effect in this contract.

2.2 Long-term Contracts

It is useful to agree a contract about what contracts may be agreed in the future. In Figure 4 we give an example standing contract specifying prices for rolls of wire mesh, fixing screws and sheets of tin roofing. The vendor agent is constrained by the standing contract to accept any purchase proposals matching the agreed criteria. The purchase proposal is a reference, by means of the contract label, to the simple purchase contract presented above.

```
customerVendorContract(customer:C, vendor:V |
   vendorBank:VB, customerBank:CB, deliveryService:DS) {

  initiates(E, oblig(V, do(X, replyTo(X, E)), T+100), T) ←
    proposeEvent(E, C, V, _).
  initiates(E, oblig(V, do(X, acceptEvent(X, E), T+100), T) ←
    proposeEvent(E, C, V,
       customerVendorContract_purchase(customer:C, vendor:V |
         vendorBank:VB, customerBank:CB,
         deliveryService:DS, item:I, price:P)) ∧
     agreedPrice(item:I, price:P).

   agreedPrice(item:wiremesh(width:10, height:10, gauge:10), price:10.00).
   agreedPrice(item:fixingscrews(gauge:5, amount:1000), price 6.99).
   agreedPrice(item:tinroofing(width:6, height:9), price 4.00).
}
```

Fig. 4. Standing Contract

The first `initiates` rule specifies that the vendor must reply to proposals (of any kind) from the customer within 100 time units. The important syntax here is the `do(...)` notation which indicates that the agent must bring about an event satisfying a particular constraint. In this case the event must satisfy the `replyTo` constraint, meaning that it must be a valid reply to the accept event (i.e. an accept or a reject). The following section defines the `replyTo`, `proposeEvent` and `acceptEvent` predicates, which are common to all agents. Similarly the second `initiates` rule specifies that the vendor must accept proposals meeting the `agreedPrice` criterion.

3 Communication

In our system, events are the act of sending messages. The messages are interpreted by the agents according to a shared ontology, the domain independent part of which is described here and in section 4. Only recorded events that are relevant to a contract may progress the contract state. Real world events, such as "the car leaving the drive way", may be put in the context of a contract by an inform event (or reported event, see below) to that effect.

We require that the events are observed by all contract principals because the state of the contract depends on the history of events relevant to it. In a two party contract, this requirement is trivially satisfied when one principal sends a message to another.

Although the exact format of a message and its transport details will vary from application to application and agent society to agent society, a well-formed message should include a time-stamp, a unique message identifier, a field identifying the message to which it is a reply (if any), message sender, message receiver, a message content, context and the interaction protocol or conversation identifiers. Messages relevant to a contract should include a context field corresponding to the contract label. This information can be inferred if the received message is in reply to an earlier message that was properly context-tagged. Figure 5 gives an example representation of an accept event for the long term contract between the customer and vendor (see Figure 4).

We list some predicates that can be used in the contract language and agent code either as tests on received messages, or as constraints on messages about to be sent (this commonly occurs when an agent is obliged to `do` a communication subject to some specified constraints). The predicates are implemented in terms of constraints on the message attributes above.

contractEvent(E, C) event `E` is in the context of contract `C`.

proposeEvent(PE, X, Y, P) `PE` is a propose event from `X` to `Y` for a proposal `P`. Proposals take the form of contract labels. Propose events must specify the propose protocol, which restricts the valid replies to accept or reject events.

acceptEvent(A, PE) `A` is an acceptance event in reply to a proposal event, `PE`. Accept events must specify the propose protocol. According to the protocol rules (see `replyTo` below), there can only be one reply to a proposal, so if there are any further accept or reject events they should be ignored. The receiving agent should check the validity of the proposal event.

rejectEvent(R, PE) `R` is a reject event in reply to a proposal event, `PE`. Like acceptEvents, rejectEvents should specify the propose protocol.

informEvent(IE, X, Y, F) `IE` is an inform event from `X` to `Y` that `F` is true. `F` is normally a fluent. Only agents that are authoritative about the fluent (see subsection 4.1) may establish it in the context of the contract.

replyTo(R, E) `R` is a reply to event `E`. If `E` specifies a protocol, `replyTo` constrains `R` to be a valid response in the protocol. `R` and `E` must share the same context and must agree on the protocol attribute. `R`'s `in-reply-to` attribute must equal `E`'s message identifier.

reportEvent(RE, E) indicates that RE is a report of an actual event E. This is most useful when E is an inform event from another contract that something has been achieved. Only events which actually occur may be reported – this constraint might be enforced by the requiring event senders to digitally sign their events.

requestEvent(E, A, F) `E` is a request event for agent `A` to bring about that F is true. A successful response is an inform event that `F` is now true.

```
accept(
  time:20050121144600, identifier:cv123, in-reply-to:cv122,
  sender:sales@wiremeshRus, receiver:jak97@imperial.ac.uk,
  content:propose(
    time:20050121130100,
    identifier:cv122,
    sender:jak97@imperial.ac.uk,receiver:sales@wiremeshRus,
    content:customerVendorContract(
      customer:jak97@imperial.ac.uk, vendor:sales@wiremeshRus,
      vendorBank:finance@bank1, customerBank:finance@bank2,
      deliveryService:deliver@pforce),
    protocol:propose),
  protocol:propose).
```

Fig. 5. Possible representation of an accept event

4 Contract Evaluation

An agent may have many contracts active at the same time. It is important to be able to consider the contracts independently of each other (for example to determine a contract's state), and also their combined effect (for example when outsourcing goals). For this reason we define a meta-interpreter predicate, `selon`[2], which evaluates queries relative to a specified contract. Subsection 4.3 describes how the individual contract effects are combined into the agent's belief store.

Figure 6 shows the core of the meta-interpreter. The first parameter is the contract label, and the second is the formula to be evaluated. The agent can now query what obligations are current with respect to a contract by asking `selon(C, holdsAt(oblig(A, G, DL), Now))` where `Now` is a time point representing the current time and `C` is the label of an active contract.

The symbols `not`, $\wedge$ and $\leftarrow$ are overloaded. Where they occur in a functional context (in the second argument to `selon`), they should be read as functional

[2] From the French, *selon*, meaning "according to"

```
selon(C, happens(E, T)) ← happens(E, T) ∧ contractEvent(E, C).
selon(C, happens(start, T)) ← happens(E, T) ∧
  initiates(E, activeContract(C), T).
selon(C, R) ← contractClause(C, R ← S) ∧ selon(C, S).
selon(C, P ∧ Q) ← selon(C, P) ∧ selon(C, Q).
selon(C, not P) ← not selon(C, P).
selon(C, holdsAt(F, T)) ← selon(C, happens(E, T1)) ∧ T1<T ∧
  selon(C, initiates(E, F, T1)) ∧ selon(C, not clipped(T, F, T1)).
selon(C, clipped(T0, F, T1)) ← selon(C, happens(E, T)) ∧ T0≤T ∧ T<T1 ∧
  selon(C, terminates(E, F, T)).
```

Fig. 6. Contract meta-interpreter

terms; where they occur in a logical context (as part of the definition of `selon`) they should be read as logical connectives. Further meta-interpreter rules are presented below defining the concepts of authoritative agents, reported events and obligation fulfilment and violation.

4.1 Authoritative Agents

In the simple purchase contract, the vendor was authoritative for the invoice number. We capture this authority with an extension to the meta-interpreter:

```
selon(C, initiates(E, F, T)) ←
  selon(C, authoritative(X, F)) ∧ informOrReportedEvent(E, X, F).

selon(C, terminates(E, G, T)) ← selon(C, incompatible(F, G)) ∧
  selon(C, authoritative(X, F)) ∧ informOrReportedEvent(E, X, F).

informOrReportedEvent(E, X, F) ← informEvent(E, X, _, F) ∨
  (reportEvent(E, I) ∧ informEvent(I, X, _, F)).
```

In the simple purchase contract, the delivery agent is authoritative for the delivery fluent. The delivery agent is not a principal of the contract, however, so in order for any delivery notification to have effect, it must be reported by one of the principals (in this case the customer or the vendor). Direct or indirect reporting of an inform event from the authoritative agent is deemed to be a valid contract event by virtue of the last rule.

We have borrowed the `incompatible` predicate, which states which fluents must be terminated in response to one being initiated, from the original event calculus[10].

4.2 Obligation Fulfilment and Violation

We adopt a similar semantics to Dignum et al.[6] with respect to deadlines. An obligation is fulfilled if the deadline has not yet expired. If the obligation was to achieve a state of affairs represented by a fluent fulfillment has to have been notified by an event that initiates the contract fluent. Where it was a more direct obligation to bring about an event characterized by a constraint, that the event has occurred is checked by showing that the constraint is now satisfied.

An obligation is violated if the deadline has elapsed and it has not been fulfilled. For simplicity's sake, we omit rules allowing a violation to be repaired (by meeting its sanction). We need three meta-interpreter rules to capture this: two for `achieve` and `do` fulfilment and one for violation.

```
selon(C, initiates(E,
  fulfilled(oblig(X, achieve(F), DL)), T)) ←
      selon(C, holdsAt(oblig(X, achieve(F), DL), T)) ∧
      T<DL ∧ selon(C, initiates(E, F, T)).
selon(C, initiates(E,
  fulfilled(oblig(X, do(E, Constraint), DL)), T)) ←
    selon(C, holdsAt(oblig(X, do(E, Constraint), DL), T))
    ∧ T<DL ∧ selon(C, Constraint).
selon(C, violated(oblig(X, G, DL), T)) ←
  selon(C, holdsAt(oblig(X, G, DL), T)) ∧ DL<T ∧
  not selon(C, holdsAt(fulfilled(oblig(B, G, DL)), T)).
```

4.3 Imported Fluents

Event calculus is used not only within the contract language definition, but also by the agent at the top-level to manage its beliefs. We need some rules to model that certain contracts have effects on the agent society outside of the contract itself. An example of this is the simple purchase contract which concludes with the transfer of ownership of the item from the vendor to the seller: reasoning solely with respect to the purchase contract will not allow the agent to realise that it does not own the item after selling it in the future. Since we need to track the ownership changes over the course of several contracts, we pool the ownership fluent into the agent's own belief store:

```
initiates(E, F, T) ← importedFluent(F) ∧
  holdsAt(activeContract(C), T) ∧ selon(C, initiates(E, F, T)).

terminates(E, F, T) ←  importedFluent(F) ∧
  holdsAt(activeContract(C), T) ∧ selon(C, terminates(E, F, T)).

importedFluent(owns(_, _)).
importedFluent(activeContract(_)).
```

The `importedFluent` predicate selects which contract fluents should be imported into the agent's belief store. `activeContract` is a fluent predicate indicating which contracts are active. Marking it as an imported fluent allows contracts to spawn sub-contracts.

5 Agent Architecture

We now describe an agent architecture in the style of AgentSpeak(L) to enable agents to respond to events related to all their active contracts in a timely fashion. We give a brief introduction to a simplified version of AgentSpeak(L), and then propose a plan library for the customer agent that will allow it to make use of the standing and purchase contracts. AgentSpeak(L) is chosen as a basis because it has a well understood operational semantics and there are available implementations such as [16] and [4].

5.1 AgentSpeak(L)

An AgentSpeak(L) agent architecture can be viewed as multi-threaded event-triggered interruptible logic programming system [11].

There are two kinds of events, belief updates and new goal events. Belief updates are represented as `+b` or `-b` depending on whether the particular belief, `b`, is now true or false. Belief events model the changes in the environment as perceived by the agent. New goal events are represented as `+!g`, where `g` is the goal to achieve.

At the beginning of the agent cycle, the agent picks an event to handle from the set of unhandled events. The plan library is consulted to see if there are any plans that are triggered by the event. Each plan in the plan library has the syntax: `event:condition <- actions`.

For example, `+temperature(T) : T > 90 <- switch(heater, off).` is a plan from an environmental control agent. The plan is relevant to changes in temperature, and applicable when the temperature rises beyond 90 degrees. The action is to switch the heater off.

If the `event` in the head of the plan unifies with the selected event, the plan is said to be relevant. The `condition` is a formula in terms of the current beliefs of the agent and acts as a guard: the relevant plans whose condition formula evaluates to true are said to be applicable. Finally, one plan is selected from the applicable plans and an intention is created to monitor it.

The agent then picks an intention to execute, which involves executing the plan body (`actions`) one step at a time. A step may be either a physical action, an achieve goal (written `!goal`) or a test (written `?test`).

Goal achievement is handled by suspending the intention and adding a new goal event to the set of unhandled events. Future agent cycles will pick up the new goal event, and look in the plan library (as before) for an applicable plan to achieve it. The plan is then stacked on top of the intention that issued the achieve goal action, so that once the goal has been achieved, execution of that intention may continue.

Tests are queries to the agent belief store, and result in a set of variable assignments which are substituted into the remaining plan steps.

5.2 Extensions to AgentSpeak(L)

We extend the AgentSpeak(L) in the following ways:

– Plans may include belief update steps, of the form `+b` or `-b`. This effects the belief store of the agent in a similar way to Prolog's assert and retract.
– An agent may have an initial set of desires, which can be selected and posted as new goal events.
– In the example plan libraries below, we have also included some Prolog style horn clauses to ease readability. Since these definitions can be folded directly into the AgentSpeak(L) rules, they do not affect the operational semantics.
– "Fire and forget" goal execution, written `!!goal`. Instead of stacking the plan for the goal on top of the existing intention, create a separate intention for the achievement of the goal. This is useful when the agent requires simply to start off a process to achieve a goal, but not to wait for its achievement.

– The textually first applicable plan is selected if there is a choice and the agent commits to that plan.
– We implement the following physical actions:
  `notify` sends a message to all the principals of a given contract (see section 3), and logs it as a communication event.
  `waitReply` waits for a reply to a given notification message to be received, subject to a timeout.
  `waitContractEvent` waits for an event that is relevant to the specified contract. This is either an incoming communication event, or the lapsing of any of the current obligations' deadlines.
  `fail` abandons an executing plan and marks it as failed.

5.3 Meta-information about Contracts

Instead of writing a set of plans to address specific contracts, such as `customerVendorContract`, we can write plans that address a general class of standing and purchase contracts. We do this by abstracting common behaviour into agent-specific meta-information about the contracts. We define a binary relation `isa` which is true iff a particular contract belongs to a more general class of contracts. The schematic rules below we say that `customerVendorContract_purchase` as an instance of `purchaseContract` and `customerVendorContract` as an instance of `standingContract` that can be used to create new `customerVendorContract_purchase` contracts, so long as the item and price information match up with the agreed prices in the standing contract.

```
customerVendorContract_purchase($\overline{X}$) isa purchaseContract($\overline{X}$).
customerVendorContract($\overline{Y}$) isa standingContract(PC) :-
   PC isa purchaseContract($\overline{Y}$, item:I, price:P)
   selon(customerVendorContract($\overline{Y}$), agreedPrice(item:I, price:P)).
```

It is also useful to know when a contract is complete. This is dependent on the specific type of contract. For example, the standing contract above is open ended - it is never completed, whereas the purchase contract ends successfully with ownership of the item. `complete` is a binary predicate, first argument is the contract and the second argument is the time of evaluation.

```
complete(PC, T) :-
   PC isa purchaseContract(customer:C, _, item:I, _, _, _, _),
   selon(PC, holdsAt(owns(owner:C, item:I), T)).
```

5.4 Plan Library for Contract Execution

We now describe a plan library for executing arbitrary contracts. For each active contract, `C`, the agent must ensure that there is an intention to abide by it, by invoking a plan for the goal `monitor(C)`.

In the case of the vendor, we assume an initial desire to abide by their standing contract, which will result in the goal to `monitor` it. However, as this standing contract does impose any obligations on the customer agent, it is not necessary for that agent to actively monitor it. As we shall see, the customer may instead make use of the contract to achieve an ownership goal by creating an active purchase sub-contract that it will monitor.

```
+!monitor(C): now(Now) & complete(C, Now) <- true.
+!monitor(C): now(Now) & selon(C, holdsAt(oblig(Self, G, DL), Now)) &
              not(observed(C,oblig(Self, G, DL)))
<- +observed(C,oblig(Self, G, DL)) ; !G by DL in C ;
     !monitor(C).
+!monitor(C): now(Now) & selon(C, holdsAt(fulfilled(Oblig), Now)) &
              not(observed(C,fulfilled(Oblig))
<- +observed(C,fulfilled(Oblig)) ; !obligFulfilled(C, Oblig);
     !monitor(C).
+!monitor(C): now(Now) & selon(C, violated(Oblig, Now)) &
              not(C,observed(violated(Oblig)))
<- +observed(C,violated(Oblig)); !obligViolated(C, Oblig) ;
    !monitor(C).
+!monitor(C): true <- waitContractEvent(C) ; !monitor(C).
```

The conditions of the above plans query the state of the contract using the `selon` predicate. The `now` predicate gives the current time. When the contract is complete, as defined by the `complete` predicate in the contract meta-information, the first rule is applicable and the execution plan terminates.

The second rule states that if there is a new obligation on the agent, a goal of the form `G by DL in C` is posted. This goal event will be handled by other plans in the plan library (see customer and vendor agent's plan libraries below). When the plan to achieve the goal completes, the `monitor(C)` goal is reposted to carry on contract execution.

The third and forth rules monitor the contract for obligation fulfilment and violation. `obligFulfilled` and `obligViolated` goals are posted, which may be handled elsewhere in the agent's plan library to keep track of, for example, the reliability and reputation of the contract participants.

The last rule states that if the contract is not yet complete, the agent waits for a communication event or for the earliest outstanding obligation deadline to lapse before consulting the contract again. Although the condition of the last plan is always true, our plan selection function selects the textually first applicable plan.

5.5 Plan Library for Customer Agent

The role of the customer agent is to respond to desires to own an item. These desires are manifested by achievement goals, which gives rise to intentions to satisfy them. We show how the agent may make use of standing contracts (or other means of achieving ownership) in an example plan library.

```
+!owns(owner:Self, item:I) :
    now(Now) & not(holdsAt(owns(owner:Self, item:I), Now)) &
    holdsAt(activeContract(SC), Now) &
    SC isa standingContract(PC) &
    PC isa purchaseContract(customer:Self, vendor:V, item:I, price:P,
            vendorBank:VB, customerBank:SelfBank, deliveryService:DS) &
    reliable(V), reliable(DS) &
    fairPrice(item:I, price:P)
```

```
 <- ?proposeEvent(Proposal, Self, V, PC) ;
    notify(SC, Proposal) ;
    waitReply(Reply, Proposal, Now + 100) ;
    !enact(Reply, Proposal).

+!enact(timedOut, Proposal) : true <- +noResponseTo(Proposal).
+!enact(Reply, Proposal) : rejectEvent(Reply, Proposal) &
    rejectEvent(Proposal, Self, V, _) <- +rejected(Proposal).
+!enact(Reply, Proposal) : acceptEvent(Reply, Proposal) &
    proposeEvent(Proposal, Self, V, PC) <- !monitor(PC).
```

The plan above is applicable to the goal of achieving ownership of a particular item, if the agent does not already own it, and there is an agreed standing contract mandating an acceptable price for the item with a (believed) reliable vendor and delivery service. The plan body constructs a proposal event and sends it to the vendor in the context of the standing contract, waits for a reply and then acts on that reply. The vendor agent is obliged to respond with an accept event within 100 time units, and should they fulfill that obligation the resulting purchase contract will be monitored by the customer. If no response comes in time, or the proposal is rejected, a belief to that effect is stored effecting the customer's future reliability estimate of the vendor.

There is only one possible obligation on the customer arising from the purchase contract, and that is to pay for the item. We make the simplifying assumption that the customer has enough money in his account:

```
+!achieve(paid(payer:Self, payee:V, price:P, reference:R)) by DL in PC :
    now(Now) & holdsAt(activeContract(BC), Now) &
    BC isa bankContract(customer: C, bank:B)
<- ?requestEvent(Request, SelfBank, paid(payer:C, payee:V, price:P,
      reference:R)) ;
   notify(BC, Request) ; waitReply(Reply, Request, Now + 100);
   ?reportEvent(Report, Reply);
   notify(PC, Report).
```

After instructing the bank to transfer the money (in the context of the contract between the customer and their bank), the customer waits for an acknowledgement that this has been done and forwards it to the vendor in the context of the purchase contract. It is this reported event that causes the paid fluent to become established, and consequently for the customer to have fulfilled the payment obligation to the vendor (see section 4.1).

5.6 Plan Library for Vendor Agent

The following plan library enables the vendor to accept and reject proposals for purchase contracts. If the vendor is obliged to accept it, then by the generic contract execution plan library, a goal will be posted to of the form `do(X, acceptEvent(X, E)) by DL in SC` which is handled by the plan below.

```
+!do(X, acceptEvent(X, E)) by DL in SC: proposeEvent(E, C, Self, Proposal)
<- ?acceptEvent(X, E) ; notify(SC, X) ; !!monitor(Proposal).
```

The condition of this plan extracts the proposed contract from the content of the message, and the plan body constructs an accept event and sends it the customer in the context of the standing contract. A separate intention is then created to monitor the proposed contract. The plan for rejecting a proposal is similar, except no intention is created to monitor the proposed contract.

Now we consider that the standing contract also obliges the agent to reply to a proposal even if it is not obliged to accept it. We define two auxiliary predicates `obligedToAccept` which is true iff the vendor is obliged to accept the proposal, and `shouldAccept` which is true iff the it is in the vendor's interest to accept the proposal. `shouldAccept` includes checks like there is available stock, taking into account already committed stock, and that the proposed price of the item is at least twice the cost price.

```
obligedToAccept(SC, E, T) :-
    selon(SC, holdsAt(oblig(Self, do(X, acceptEvent(X, E)), DL), T)).

shouldAccept(Proposal, T) :-
    Proposal isa purchaseContract(customer:C, vendor:Self | price:P,
              vendorBank:SelfBank, customerBank:CB, deliveryService:DS),
    costPrice(item:I, price:CostPrice),
    warehouse(item:I, availability:Warehouse),
    committed(item:I, level:Committed),
    Warehouse - Committed > 0,
    P >= CostPrice * 2.
```

The following two plans make use of the predicates to decide whether to accept or reject the proposal. The plan bodies are simply goals to accept or reject which will be handled by the plans as the start of this subsection.

```
+!do(X, replyTo(X, E) by DL in SC: proposeEvent(E, C, Self, Proposal) &
    now(Now) & shouldAccept(Proposal, Now) &
    not(obligedToAccept(SC, E, Now))
<- +!do(X, acceptEvent(X, E)) by DL in SC.

+!do(X, replyTo(X, E) by DL in SC: proposeEvent(E, C, Self, Proposal) &
    now(Now) & not(shouldAccept(Proposal, Now)) &
    not(obligedToAccept(SC, E, Now))
<- +!do(X, rejectEvent(X, E)) by DL in SC.
```

There are two obligations that may result on the vendor during execution of the purchase contract. The first is an obligation to announce an invoice number, and the second is to arrange for delivery of the item.

```
+!achieve(value(invoice-no, _)) by DL in PC : true
<- -invoiceNo(Last) ;
    ?New is Last + 1; +invoiceNo(New) ;
    ?informEvent(E, Self, value(invoice-no, New)) ; notify(PC, E).
```

To achieve a fresh value for the invoice number, the vendor increments a belief atom, `invoiceNo`, and then creates an inform event asserting its value to send to the customer in the context of the purchase contract, `PC`.

```
+!achieve(delivered(item:I, destination:C, invoice-no:R)) by DL in PC :
   PC isa purchaseContract(customer:C, vendor:Self | price:P,
            vendorBank:SelfBank, customerBank:CB, deliveryService:DS) &
   now(Now) & holdsAt(activeContract(DC), Now),
   DC isa deliveryContract(customer:V, deliveryService:DS)
<- ?requestEvent(Request,DS,delivered(item:I,destination:C,invoice-no:R)) ;
   notify(DC, Request) ; waitReply(Reply, Request, Now + 100) ;
   ?reportEvent(Report, Reply) ; notify(PC, Report).
```

The vendor agent has no built-in capability to achieve delivery, so it must make use of a third party. The above plan checks that its has an active contract, DC, with the delivery service `DS` mentioned in the purchase contract (which we assume it will have).

The vendor must create a request event to achieve the delivery and sent it to DS in the context of `DC`. If successful, the delivery service will reply with an inform that the item has been delivered, which is then forwarded by the vendor to the customer in the context of the purchase contract, thus fulfilling the vendor's obligation to deliver the item.

6 Related Work

Our architecture shares the concepts of plan library, beliefs, intentions with AgentSpeak(L). The addition of contracts not only reifies the concept of obligations, but also extends the built-in behaviour of the agent by allowing it to outsource goals that it cannot achieve itself.

In Agent0[13], agents are programmed by specifying a set of capabilities (commitment rules). Instead of building the commitment rules directly into the agent, our architecture allows these rules to be specified in the contract in the form of event calculus initiates and terminates rules.

Verharen's cooperative information agents [14] [15] are based on the language action perspective. The architecture specifies three main categories of activities: tasks (plans to achieve tasks organised with dependencies between the tasks), transactions (message sequences organised with temporal ordering constraints), and contracts, which are represented as deontic state machines of transaction transitions. Our system does not mandate such a conceptual break down, rather we envisage that higher level contract languages may be translatable into our simpler event calculus syntax.

Social integrity constraints[1] can be used to specify and verify agent interaction protocols. The constraints express expectations on events that ought to (or ought not to) occur given the occurrence of a previous triggering event. Each expectation has an associated priority such that those with higher priority are more ideal. The concept is related to obligations, but their focus is different: social integrity constraints focus essentially on constraining the event history, whereas obligations focus on constraining a particular agent behaviour. It would be an interesting to attempt the expression of the one in terms of the other.

Artikis et al[2] describe a system for animating and specifying computational societies. The system takes a global perspective, and so in order to make inferences about the state of the society all the events relevant to it must be known.

This is in contrast to our work, where conclusions are reached relative to each contract rather than the society as a whole.

Kollingbaum and Norman describe a system of supervised interaction[8] where agents are supervised by a third party called an authority. The authority registers contracts between the agents, witnesses the communications between the agents and enforces the norms specified in the contracts. Our system does not require this infrastructure, although it does admit a logging agent should the particular situation demand it. Furthermore the agents themselves are responsible for enforcing the contractual norms.

It is important that the contract is carefully constructed so that prohibitions do not completely prevent the fulfillment of obligations. It is feasible to statically check contracts for these kinds of potential conflicts[3]. The NoA architecture[9] solves conflicts by prioritising permissions over prohibitions — obligation consistency is determined by considering the action effects of plans to handle the obligation. If all plans include actions that are prohibited or interfere with other obligations, the obligation is found to be inconsistent and is not adopted.

Conflicts between obligations and desires may also emerge, and if so conflict resolution will be important. The BOID architecture[5] describes a method of resolving these conflicts. Beliefs, obligations, intentions and desires are represented as separate components with feedback loops between them. Each component builds extensions (closure under logical consequence) of their propositional theories, and conflicts between the components are resolved by prioritising the components one over the other.

7 Conclusion

Contracts are a powerful and high level approach to programming agent behaviour. Furthermore, specifying the contractual relationships between agents separately to the agents' capabilities is not only good software engineering, because concerns are separated, but also facilitates analysis and verification since the contracts are represented in a formal language, the event calculus. Event calculus is especially suitable for contract language representation because the semantics are unambiguous and, given a reliable log of events, the conclusions derived cannot be disputed.

Finally, our agent architecture provides a simple, powerful, extensible means to implement *passive* (by monitoring fulfilment and violation of obligations), *reactive* (by reacting to new obligations), *proactive* (by taking advantage of contracts to oblige other agents) and *opportunistic* (by accepting proposals that are in the agent's interest, but not necessarily obliged to accept) behaviours.

References

1. M. Alberti, D. Daolio, P. Torroni, M. Gavanelli, E. Lamma, and P. Mello. Specification and verification of agent interaction protocols in a logic-based system. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 72–78, New York, NY, USA, 2004. ACM Press.

2. A. Artikis, J. Pitt, and M. Sergot. Animated specifications of computational societies. In C. Castelfranchi and L. Johnson, editors, *Proceedings of Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1053–1062. ACM Press, 2002.
3. A. K. Bandara, E. C. Lupu, and A. Russo. Using event calculus to formalise policy specification and analysis. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, page 26. IEEE Computer Society, 2003.
4. R. H. Bordini and J. F. Hübner. Jason - A Java-based agentSpeak intepreter used with saci for multi-agent distribution over the net. `http://jason.sourceforge.net`, 2005.
5. J. Broersen, M. Dastani, J. Hulstijn, Z. Huang, and L. van der Torre. The BOID architecture: conflicts between beliefs, obligations, intentions and desires. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 9–16. ACM Press, 2001.
6. F. Dignum, J. Broersen, V. Dignum, and J.-J. Meyer. Meeting the deadline: Why, when and how. In *Proceedings of the 3rd Conference on Formal Aspects of Agent-Based Systems (FAABS III)*, 5 2004.
7. V. Dignum, J.-J. Meyer, F. Dignum, and H. Weigand. Formal specification of interaction in agent societies. In *Formal Approaches to Agent-Based Systems*, number 2699 in LNAI. Springer, 2002.
8. M. J. Kollingbaum and T. J. Norman. Supervised interaction: creating a web of trust for contracting agents in electronic environments. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 272–279. ACM Press, 2002.
9. M. J. Kollingbaum and T. J. Norman. Norm consistency in practical reasoning agents. In *Proceedings of PROMAS Workshop on Programming Multiagent Systems, AAMAS 2003*, 2003.
10. R. A. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(4):319–340, 1986.
11. A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. V. Velde and J. W. Perram, editors, *Agents Breaking Away (LNAI 1038)*, pages 42–55. Springer-Verlag, 1996.
12. M. Shanahan. The event calculus explained. In M.J.Wooldridge and M.Veloso, editors, *Artificial Intelligence Today*, volume 1600 of *LNAI*, pages 409–430. Springer Verlag, 1999.
13. Y. Shoham. Agent0: A simple agent language an its interpreter. In *Proc. of AAAI-91*, pages 704–709, Anaheim, CA, 1991.
14. E. Verharen and F. Dignum. Cooperative Information Agents and communication. In P. Kandzia and M. Klusch, editors, *Cooperative Information Agents, First International Workshop*, number 1202 in LNAI, pages 195–209, 1997.
15. E. Verharen, F. Dignum, and S. Bos. Implementation of a cooperative agent architecture based on the language-action perspective. In M. Singh, editor, *Intelligent Agents IV*, volume 1365 of *LNAI*, pages 31–44. 1998.
16. M. Winikoff. An agentspeak meta-interpreter and its applications. In *Proceedings of the Third international Workshop on Programming Multi-Agent Systems*, 2005.

Profiles of Behaviour for Logic-Based Agents

F. Sadri and F. Toni

Department of Computing, Imperial College London,
{fs,ft}@doc.ic.ac.uk

Abstract. In an earlier paper [6] we presented a declarative approach for agent control. In that work we described how control can be specified in terms of *cycle theories*, which define declaratively the possible alternative behaviours of agents, depending on their internal state and (their perception of) the external environment in which they are situated. This form of control has been adopted for logic-based *KGP agents* [8, 2]. In this paper we show how using this form of control specification we can specify different *profiles* of agents, how they would vary the *behaviour* of agents and what advantages they have with respect to factors in the application and in the environment, such as time-criticality.

1 Introduction

In an earlier paper [6] we described how to specify control of agents via cycle theories. The approach is based on representing and reasoning with preferences and allows flexible control of the operations of agents. This takes the control beyond a fixed one-size-fits-all approach and allows the operations of the agents to be chosen dynamically given the circumstances of the environment, the state of the agent and its preferences.
Cycle theories have been adopted as the means of control in the KGP agent model [8, 2] developed within the SOCS project [1]. KGP is a modular logic-based model developed to cater for the challenges of open global computing environments. It relies upon a collection of capabilities utilised within transitions controlled by cycle theories. All the components are defined within computational logic, some using abductive logic programming and others using logic programming with preferences. The capabilities are designed to provide functionalities such as planning, reactivity, temporal reasoning and goal decision, all of which have been envisaged useful, maybe even necessary, for coping and adapting in a dynamic open environment. The KGP model has been implemented within the PROSOCS platform [12].
The behaviour of KGP agents can be seen as the sequence of transitions or operations they perform, and this sequence is determined by the agents' cycle theories. Thus by varying the cycle theory one can vary the behaviour of the agent. We have explored a number of such variations resulting in different profiles of behaviour. In an earlier paper [6] we briefly mentioned three, the focussed, careful and impatient profiles. In this paper we detail the first two. We characterise them formally, show how to design cycle theories that achieve them and discuss their advantages depending on the features of the environment and application domains. Other profiles are described in [1].

[1] `http://lia.deis.unibo.it/Research/Projects/SOCS/`

The motivation for this work is threefold: 1) to explore the degree of heterogeneity that can be achieved by varying cycle theories; 2) to explore the advantages of different profiles of behaviour with respect to different parameters such as the dynamic nature of the environment and the time-critical nature of applications; 3) to explore how such analysis can provide guidelines for implementers who use the PROSOCS platform.

Environments and circumstances in which agents have to function can vary. Some environments can be fairly static and predictable, while others can be highly dynamic and unpredictable. Agents may or may not have strict deadlines for their activities, and agents' resources may be limited, thus constraining what they can do, or they may have few resource restrictions. What interests us in this paper is to explore what profiles of behaviour would be advantageous in what type of environment and under what circumstances. Moreover, we would like to explore how to define such profiles by varying the control strategies of agents defined via cycle theories.

The paper is organised as follows. In Section 2 we present two examples to motivate the careful and focussed profiles. In Sections 3 and 4 we describe the necessary background to our work. In Section 5 we describe the careful and the focussed profiles in detail, show the characteristics of cycle theories that provably achieve these profiles, and discuss the pros of the two behaviour profiles. In Section 6 we conclude.

2 Motivating Examples

In this section we motivate, in the context of concrete examples, the two profiles we will study and formally define later, in Section 5.

2.1 Careful profile

Intuitively an agent endowed with this profile frequently re-examines its commitments to ensure that he honours only those that are feasible and necessary and he is not encumbered by any infeasible or unnecessary commitments. The advantage of such a profile is evident in a dynamic, unpredictable environment.

Consider an agent c who has sent its registration form to a conference $conf05$ and thus believes that it has registered for the conference. But it now wishes not to be registered at the conference. It sets itself this goal, and plans for it by generating an action to cancel his registration at $conf05$. Suppose the agent knows that :

> *If it observes that the deadline for cancellation for a conference has reached and it expects to cancel its registration at the conference then it should contact its bank and tell them to stop its credit card payment to the conference.*

Suppose before it has a chance to execute the action of cancellation of its registration it receives a message from the conference secretary telling it that there was a problem with its initial attempt at registration (for example the registration form arrived corrupted) and so it is actually not registered.

An agent with the careful profile will immediately realise that there is no longer any need to cancel its registration and consequently will not contact its bank to tell them to stop the credit card payment. But, under the same circumstances, an agent with a different profile might execute the (unnecessary) acts of contacting the bank and canceling the payment.

2.2 Focused profile

An agent may attempt to plan for multiple goals at once or may plan for one goal at a time. If the agent has limited resources it may be better off trying one goal at a time because typically it may not have enough resources for achieving multiple goals, and attempting to do so would only lead to time-wasting failures. This is the motivation behind our focussed profile. An agent endowed with the focussed profile focuses on one goal at a time.
Suppose an agent has two goals, one to have a particular book and the other to have a particular CD. Suppose the book costs £10 and the CD £15, and the agent has £20 available to spend. This agent cannot achieve both its goals, because due to its financial constraints it cannot form a consistent plan that would achieve both goals. If the agent has the focussed profile it will achieve one of them but if it has any other profile it may not achieve either goal.
In the next two sections we give the background that is necessary in order to formally define the profiles and show their consequences in terms of the behaviour of agents.

3 The KGP Model of Agency

Here we briefly summarise the KGP model for agents. Formal details can be found in [8, 2]. This model relies upon

- an *internal (or mental) state*,
- a set of *reasoning capabilities*, in particular supporting planning, temporal reasoning, reactivity and goal decision,
- a *sensing capability*, allowing agents to observe their environment and actions by other agents,
- a set of *transition rules*, defining how the state of the agent changes, and defined in terms of the above capabilities,
- a set of *selection functions*, to provide appropriate inputs to the transitions,
- a *cycle theory*, for deciding which transitions should be applied when, and defined using the selection functions.

Internal state. This is a tuple $\langle KB, Goals, Plan, TCS \rangle$, where:

- KB is the knowledge base of the agent, and describes what the agent knows (or believes) of itself and of the environment. KB consists of various modules supporting the different reasoning capabilities of agents, and including KB_0, for holding the (dynamic) knowledge of the agent about the external environment in which it is situated.
- $Goals$ is the set of properties that the agent wants to achieve, each one with an associate time variable, possibly constrained by temporal constraints (belonging to TCS), defining when the goals are expected to hold.
- $Plan$ is a set of actions scheduled in order to satisfy goals. Each has an associated time variable, possibly constrained by temporal constraints in TCS, similarly to $Goals$, but defining when the action should be executed and imposing a partial order over actions in $Plan$. Each action is also equipped with the preconditions for its successful execution.

– TCS is a set of constraint atoms (referred to as *temporal constraints*) in some given underlying constraint language with respect to some structure equipped with a notion of *Constraint Satisfaction*. We assume that the constraint predicates include $<, \leq, >, \leq, =, \neq$. These constraints bind the time of goals in $Goals$ and actions in $Plan$. For example, they may specify a time window over which the time of an action can be instantiated, at execution time.

Goals and actions are uniquely identified by their associated time variable, which is implicitly existentially quantified within the overall state.
To aid revision and partial planning, $Goals$ and $Plan$ form a *tree* [2]. The tree is given implicitly by associating with each goal and action its parent. *Top-level* goals and actions are children of the root of the tree, represented by the (arbitrary) symbol $\perp$.

Reasoning capabilities. These include:

– Planning, generating a plan, if one exists in the overall state, for any given set of input goals. These plans are *partial* or *total*. A partial plan consists of (temporally constrained) sub-goals and actions. A *total* plan consists solely of (temporally constrained) actions.
– Reactivity, reacting to perceived changes in the environment by modifying $Goals$, $Plan$, and TCS.
– Goal Decision, revising the top-most level goals by adapting the agent's state to changes in its own preferences and in the environment.
– Temporal Reasoning, reasoning about the evolving environment, and making predictions about properties holding in the environment, based on the partial information the agent acquires.

Transitions. The state of an agent evolves by applying transition rules, which employ capabilities and the Constraint Satisfaction. The transitions are:

– *Goal Introduction (GI)*, changing the top-level goals, and using Goal Decision.
– *Plan Introduction (PI)*, changing $Goals$ and $Plan$, and using Planning.
– *Reactivity (RE)*, changing $Goals$ and $Plan$, and using the Reactivity capability.
– *Sensing Introduction (SI)*, changing $Plan$ by introducing new sensing actions for checking the preconditions of actions already in $Plan$, and using Sensing.
– *Passive Observation Introduction (POI)*, changing KB_0 by introducing unsolicited information coming from the environment, and using Sensing.
– *Active Observation Introduction (AOI)*, changing KB_0 by introducing the outcome of (actively sought) sensing actions, and using Sensing.
– *Action Execution (AE)*, executing actions, and thus changing KB_0.
– *State Revision (SR)*, revising $Goals$ and $Plan$, and using Temporal Reasoning and Constraint Satisfaction.

[2] In the detailed model we actually have two trees, the first containing *non-reactive* goals and actions, the second containing *reactive* goals and actions. All the top-level non-reactive goals are either assigned to the agent by its designer at birth, or they are determined by the Goal Decision capability, via the GI transition (see below). All the top-level reactive goals and actions are determined by the Reactivity capability, via the RE transition (see below). Here for simplicity we overlook the distinction amongst the two trees.

The effect of transitions is dependent on the concrete time of their application. We briefly describe SR, as it will play an important role in section 5. Informally speaking, SR revises a state by removing (i) all timed-out goals and actions, (ii) all executed actions, (iii) all goals that have become obsolete because they are already believed to have been achieved, (iv) siblings (in the tree) of goals and actions deleted in (i), and (v) all descendants (in the tree) of goals deleted in (i)-(iv). A goal or action is *timed-out* if and only if the temporal constraints TCS of the state of the agent at the time of application of SR constrain the time of the goal or action to be less than or equal to the time of application of SR. A goal is *achieved* in a state if and only if it holds according to the Temporal Reasoning capability.

Selection functions. Input to (some of the) transitions is given via selection functions, taking the current state S and time τ as input:

– *action selection function*, $c_{AS}(S, \tau)$, returning the set of actions in S to be executed by AE at time τ;
– *goal selection function*, $c_{GS}(S, \tau)$, returning the set of goals in S to be planned for by PI at time τ;
– *fluent selection function*, $c_{FS}(S, \tau)$, returning the set of properties in S to be sensed by AOI at time τ;
– *precondition selection function*, $c_{PS}(S, \tau)$, returning the set of preconditions of actions in S for which sensing actions are to be introduced by SI at time τ.

4 Cycle Theories

The behaviour of agents results from the application of transitions in sequences, repeatedly changing the state of the agent. These sequences are not fixed a priori, as in conventional agent architectures, but are determined dynamically by reasoning with declarative cycle theories, giving a form of flexible control. Cycle theories are given in the framework of Logic Programming with Priorities (LPP). For the purposes of this paper, we will assume that an *LPP-theory*, referred to as $\mathcal{T}$, consists of four parts:

(i) a low-level part P, consisting of a logic program; each rule in P is assigned a name, which is a term; e.g., one such rule, with name $n(X)$, could be
$n(X) : p(X) \leftarrow q(X, Y), r(Y)$
(ii) a high-level part H, specifying conditional, dynamic priorities amongst rules in P; e.g., one such priority rule, called $h(X)$, could be $h(X) : n(X) \succ m(X) \leftarrow c(X)$, to be read: if (some instance of) the condition $c(X)$ holds, then the rule in P with name (the corresponding instance of) $n(X)$ should be given higher priority than the rule in P with name (the corresponding instance of) $m(X)$.
(iii) an auxiliary part A, defining predicates occurring in the conditions of rules in P and H and not in the conclusions of any rule in P;
(iv) a notion of incompatibility, here given as a set of rules defining the predicate $incompatible$, e.g. $incompatible(p(X), p'(X))$, to be read: any instance of the literal $p(X)$ is incompatible with the corresponding instance of the literal $p'(X)$. We refer to the set of all incompatibility rules as I.

Any concrete LPP framework is equipped with a notion of entailment, that we denote by $\models_{pr}$. Intuitively, $\mathcal{T} \models_{pr} \alpha$ iff α is the “conclusion” of a sub-theory of $P \cup A$ which is “preferred” with respect to $H \cup A$ in $\mathcal{T}$ over any other sub-theory of $P \cup A$ that derives a “conclusion” incompatible with α (with respect to I). Here, we are assuming that the underlying logic programming language is equipped with a notion of “entailment” that allows to draw “conclusions”. In [10, 9, 7, 5, 3], $\models_{pr}$ is defined via argumentation.

Formalisation of Cycle theories. Here and in the rest of the paper, we will use notation $T(S, X, S', \tau)$ to represent application of transition T at time τ in state S, given input X, resulting in state S', and notation $*T(S, X)$ to represent that transition T can be potentially chosen as the next transition in state S, with input X.
Formally, a cycle theory $\mathcal{T}_{cycle}$ consists of the following parts.

– An *initial* part $\mathcal{T}_{initial}$, that determines the possible transitions that the agent could perform when it starts to operate. Concretely, $\mathcal{T}_{initial}$ consists of rules of the form
$$*T(S_0, X) \leftarrow C(S_0, \tau, X), now(\tau)$$
which we refer to via the name $\mathcal{R}_{0|T}(S_0, X)$. These rules sanction that, if conditions C hold in the initial state S_0 at the initial time τ, then the initial transition could be T, applied to state S_0 and input X.
– A *basic* part $\mathcal{T}_{basic}$ that determines the possible transitions following given transitions, and consists of rules of the form
$$*T'(S', X') \leftarrow T(S, X, S', \tau), EC(S', \tau', X'), now(\tau')$$
which we refer to via the name $\mathcal{R}_{T|T'}(S', X')$. These rules sanction that, after transition T has been executed, starting at time τ in the state S and resulting in state S', and the conditions EC evaluated in S' at the current time τ' are satisfied, then transition T' could be the next transition to be applied in S', with input X'. EC are called *enabling conditions* as they determine when T' can be applied after T. They also determine input X' for T', via calls to selection functions.
– A *behaviour* part $\mathcal{T}_{behaviour}$ that contains rules describing dynamic priorities amongst rules in $\mathcal{T}_{basic}$ and $\mathcal{T}_{initial}$. Rules in $\mathcal{T}_{behaviour}$ are of the form
$$\mathcal{R}_{T|T'}(S, X') \succ \mathcal{R}_{T|T''}(S, X'') \leftarrow BC(S, X', X'', \tau), now(\tau)$$
with $T' \neq T''$, which we will refer to via the name $\mathcal{P}^{T}_{T' \succ T''}$. Recall that $\mathcal{R}_{T|T'}(\cdot)$ and $\mathcal{R}_{T|T''}(\cdot)$ are (names of) rules in $\mathcal{T}_{basic} \cup \mathcal{T}_{initial}$. Note that, with an abuse of notation, T could be 0 in the case that one such rule is used to specify a priority over the *first* transition to take place, in other words, when the priority is over rules in $\mathcal{T}_{initial}$. These rules in $\mathcal{T}_{behaviour}$ sanction that, at the current time τ, after transition T, if the conditions BC hold, then we prefer the next transition to be T' over T''. The conditions BC are called *behaviour conditions* and give the behavioural profile of the agent.
– An *auxiliary part* including definitions for any predicates occurring in the enabling and behaviour conditions.
– An *incompatibility part*, in effect expressing that only one (instance of a) transition can be chosen at any one time.

Hence, $\mathcal{T}_{cycle}$ is an LPP-theory where: (i) $P = \mathcal{T}_{initial} \cup \mathcal{T}_{basic}$, and (ii) $H = \mathcal{T}_{behaviour}$.

Operational Trace. The cycle theory $\mathcal{T}_{cycle}$ of an agent is responsible for its behaviour, in that it induces an *operational trace* of the agent, namely a (typically infinite) sequence of transitions

$T_1(S_0, X_1, S_1, \tau_1), \ldots, T_i(S_{i-1}, X_i, S_i, \tau_i), T_{i+1}(S_i, X_{i+1}, S_{i+1}, \tau_{i+1}), \ldots$

such that

- S_0 is the given initial state;
- for each $i \geq 1$, τ_i is given by the clock of the system ($\tau_i < \tau_{i+i}$);
- $(\mathcal{T}_{cycle} - \mathcal{T}_{basic}) \wedge now(\tau_1) \models_{pr} *T_1(S_0, X_1)$;
- for each $i \geq 1$
  $(\mathcal{T}_{cycle} - \mathcal{T}_{initial}) \wedge T_i(S_{i-1}, X_i, S_i, \tau_i) \wedge now(\tau_{i+1}) \models_{pr} *T_{i+1}(S_i, X_{i+1})$

namely each (non-final) transition in a sequence is followed by the most preferred transition, as specified by $\mathcal{T}_{cycle}$. If, at some stage, the most preferred transition determined by $\models_{pr}$ is not unique, we choose one arbitrarily.

Normal cycle theory. In defining profiles in section 5 we take the *normal cycle theory* as a starting point. This specifies a pattern of operation where the agent prefers to follow a sequence of transitions that allows it to achieve its goals in a way that matches an expected "normal" behaviour. Basically, the "normal" agent first introduces goals (if it has none to start with) via GI, then reacts to them, via RE, and then repeats the process of planning for them, via PI, executing (part of) the chosen plans, via AE, revising its state, via SR, until all goals are dealt with (successfully or revised away). At this point the agent returns to introducing new goals via GI and repeating the above process. Whenever in this process the agent is interrupted via a passive observation, via POI, it chooses to introduce new goals via GI, to take into account any changes in the environment. Whenever it has actions which are "unreliable", in the sense that their preconditions definitely need to be checked, the agent senses them (via SI) before executing the action. Whenever it has actions which are "unreliable", in the sense that their effects definitely need to be checked, the agent actively introduces actions that aim at sensing these effects, via AOI, after having executed the original actions. The full definition of the normal cycle theory is given in the appendix.

5 Behaviour Profiles

In this section we explore how cycle theories can be used to specify different profiles of behaviour. We concentrate on two profiles, the *careful* and the *focussed*.

In the careful profile the behaviour of the agent is such that it would re-examine its commitments in terms of its goals and plans frequently to discard those that are no longer needed or have become infeasible. Intuitively, this profile would be suitable for a changing environment that intervenes in the agent's operations, and the frequent "self-examination" of the agent can help it avoid being occupied with unnecessary activity or activity which is bound to fail. It also ensures that the agent's operations are not hindered by superfluous items in the state and that reactive rules will not be triggered unnecessarily by goals/actions that are timed-out and not achieved/executed.

With the focussed profile the agent concentrates on one (top-level) goal at a time and only moves to other goals when that goal is achieved or is timed out. Intuitively this

profile is useful when the agent has goals that have become mutually unachievable. By being focussed the agent increases its chances of achieving at least some of them.
Below we proceed to define each of the two profiles by giving a formal definition in terms of trace characteristics, followed by specification of cycle theories that will induce such traces. We then proceed to prove the advantages of the profile depending on particular characteristics of the application.

5.1 Careful Profile

Definition 1 (Careful profile: trace-based characterisation). *A careful agent is an agent that will never generate an operational trace with two consecutive transitions that are different from SR.*
In fact, this condition is stronger than strictly necessary: As long as there are no redundant or infeasible goals or actions no revision would be required. However, from a pragmatic point of view, Definition 1 nevertheless provides us with an appropriate characterisation of careful agents. This is so, because *checking* whether or not a state includes redundant or infeasible goals or actions to be revised is just as costly as performing a state revision in the first place.
Our next goal is to define a class of cycle theories that are guaranteed to induce an operational trace where every other transition is an SR. As we shall see this is not as straightforward a goal as it may seem. To illustrate the difficulties and to motivate our choices (which are eventually going to overcome these difficulties), we start by attempting to define a careful cycle theory as an extension of the normal cycle theory.

The normal-careful cycle theory. There are several ways of combining cycle theories (in this case the normal cycle theory with the core rules necessary for characterising the careful profile). One option would be to take the union of the two cycle theories (which are sets of basic and behaviour rules) and then, where necessary, to introduce additional behaviour rules that determine the agent's behaviour in case of conflict between the rules stemming from the different parts. Another way, which gives the profile designer less freedom but which results in much simpler cycle theories, would be to work at the level of basic rules as far as possible and to use suitable enabling conditions to control the agent's behaviour. This is the approach we are going to follow here.
To design a careful agent, we need to ensure that basic rules expressing that *SR* should follow any other transition T get priority over any conflicting rules. Instead of using behaviour rules to this effect, we are simply going to delete such conflicting rules in the first place. Hence, we end up with the following approach:

– **Step 1:** Take the normal cycle theory as a starting point.
– **Step 2:** Remove any basic rules (in $\mathcal{T}_{basic}$) that speak about two consecutive transitions both of which are different from *SR*.
– **Step 3:**Add the following basic rule (to $\mathcal{T}_{basic}$) for each T different from *SR:*

$$\mathcal{R}_{T|SR}(S', \{\}) : *SR(S', \{\}) \leftarrow T(S, X, S', \tau)$$

Note that there cannot be any enabling conditions in this kind of new rule: *SR* needs to be enabled under *any* circumstances. Note also that Step 3 might re-introduce rules which already belong to $\mathcal{T}_{basic}$. This causes no theoretical or practical problem. We thus end up with the following *normal-careful cycle theory:*

– $\mathcal{T}_{initial}$ is as for the normal cycle theory.
– $\mathcal{T}_{basic}$ consists of the above rules of the form $\mathcal{R}_{T|SR}$ and of the following rules:

$$\mathcal{R}_{SR|PI}(S', Gs) : *PI(S', Gs) \leftarrow SR(S, \{\}, S', \tau'), Gs = c_{GS}(S', \tau), Gs \neq \{\}, now(\tau)$$
$$\mathcal{R}_{SR|GI}(S', \{\}) : *GI(S', \{\}) \leftarrow SR(S, \{\}, S', \tau'), Gs = c_{GS}(S', \tau), Gs = \{\}, now(\tau)$$

$\mathcal{T}_{basic}$ does not contain any other rules, because all the remaining basic rules in the normal cycle theory speak about transitions that should follow transitions other than *SR* and these are fixed for the careful profile.
– $\mathcal{T}_{behaviour}$ is empty. Indeed, it turns out that also all of the rules in $\mathcal{T}_{behaviour}$ in the normal cycle theory are *redundant*, because they speak about what to do after a transition other than *SR*.

In summary, the normal-careful cycle theory will force an agent to alternate between *SR* and *PI* or *GI* (depending on whether there are currently goals to plan for or not). Such an agent would be careful, but not very useful. Below we improve the cycle theory to overcome this inadequacy.

The core-careful cycle theory. We improve the normal-careful cycle theory by adding that every transition except *SR*, itself, should be enabled after *SR*. Thus, $\mathcal{T}_{basic}$ in the *core-careful cycle theory* contains, in addition to the basic rules in the normal-careful cycle theory, the following rules:

$$\mathcal{R}_{SR|RE}(S', \{\}) : *RE(S', \{\}) \leftarrow SR(S, \{\}, S', \tau)$$
$$\mathcal{R}_{SR|AE}(S', As) : *AE(S', As) \leftarrow SR(S, \{\}, S', \tau'), As = c_{AS}(S', \tau), As \neq \{\}, now(\tau)$$
$$\mathcal{R}_{SR|SI}(S', Ps) : *SI(S', Ps) \leftarrow SR(S, \{\}, S', \tau'), Ps = c_{PS}(S', \tau), Ps \neq \{\}, now(\tau)$$
$$\mathcal{R}_{SR|AOI}(S', Fs) : *AOI(S', Fs) \leftarrow SR(S, \{\}, S', \tau'), Fs = c_{FS}(S', \tau), Fs \neq \{\}, now(\tau)$$
$$\mathcal{R}_{SR|POI}(S', \{\}) : *POI(S', \{\}) \leftarrow SR(S, \{\}, S', \tau)$$

The following proposition states the *correspondence* between the *core-careful cycle theory* and the (trace-based characterisation of the) careful profile given in Definition 1:

Proposition 1 (Careful profile). *The core-careful cycle theory induces the careful profile of behaviour: Any agent using this cycle theory will never generate an operational trace with two consecutive transitions that are different from SR.*

Proof. This follows immediately from the fact that the basic part of the cycle theory forces an SR after every other type of transition, and there is exactly one basic rule to determine the follow-up of any transition different from *SR*.

Other careful cycle theories The two careful cycle theories we have considered so far are just two examples; there is a range of cycle theories that conform to the careful behaviour profile. Our second example, the core-careful cycle theory is the most general cycle theory conforming to the careful profile.
For concrete applications, we may wish to combine the features of careful behaviour with other more specific features. We can construct a careful cycle theory of our choice by taking the core-careful cycle theory as a starting point and then imposing additional behaviour constraints using the following means:
– strengthening the enabling conditions in basic rules that determine the follow-up transition for an SR;
– deleting basic rules that determine the follow-up transition for an SR;

– adding any kind of behaviour rules;
– deleting rules that have become redundant due to other changes.

Note, however, that we *cannot* add any enabling conditions to the basic rules that state that *SR* has to follow any other transition. Otherwise, the resulting cycle theory cannot be guaranteed to conform to the careful profile of behaviour anymore. We also cannot delete such a rule, unless it has already become redundant due to other changes in the cycle theory. On the other hand, we do have complete freedom with respect to the behaviour rules we might wish to add, because the basic rules never admit any conflict as to what transition to choose after a transition different from *SR* in the first place.
Clearly, any such careful cycle theory will also induce the careful profile of behaviour in the sense of Proposition 1.

A property of the careful profile. Informally, under certain circumstances:

– Careful agents will never generate a reaction via the reactivity transition to timed-out unachieved goals or timed-out unexecuted actions.
– Careful agents will never generate a reaction via the reactivity transition to actions that may not be timed out yet but which are unexecuted and are no longer necessary.

More formally:

Theorem 1. *The following will never contribute to the generation of a reaction (i.e. an action in $Plan$ or goal in $Goals$) via the RE transition:*

1. *a timed-out unexecuted action,*
2. *a timed-out unachieved goal,*
3. *an unexecuted action whose execution is no longer needed, i.e.*
   (a) *with an ancestor which has already been achieved, or*
   (b) *with a sibling that has been timed-out, or*
   (c) *with an ancestor which has been timed-out,*

provided that no action and no goal is timed out between an SR transition and its immediate successor if that is an RE transition.

Proof. Let the assumption hold that no action and no goal is timed out between an SR transition and its immediate successor if that is an RE transition. Suppose a careful agent applies RE in a state $S = \langle KB, Goals, Plan, TCS \rangle$. Then by Definition 1, because SR must have been applied in the state immediately prior to S, no action or goal of the type specified in 1–3, above exists in state S. Therefore no such action or goal could possibly contribute to the generation of any reaction by RE.

5.2 Focussed Profile

In the *focussed* profile of behaviour an agent does not plan for more than one top-level goal at a time. More specifically, a focussed agent remains committed to a goal amongst its top-level goals until

– that goal has been successfully achieved, or
– that goal has become infeasible, or
– that goal is not preferred by the Goal Decision capability anymore, when invoked by the *GI* transition, or

– that goal has an empty plan in the state.[3]

The advantages of the focussed profile come into effect in highly time-critical domains as well as domains where an agent has several goals with mutually incompatible plans. In such situations, a focussed agent can be expected to achieve, at least, some goals, whereby an unfocussed agent may fail completely. This applies, in particular, to agents that have a preference for total planning. By concentrating planning on a single goal at a time, a focussed agent is likely to be faster and it will also avoid wasting computing resources over incompatible plans for other goals.

Formally, the focussed profile has the following characteristic: A focussed agent, under no circumstances, will generate an operational trace that includes a state with two distinct top-level goals with children, neither of which is either achieved or infeasible. Here, a goal G is called *feasible* iff neither itself nor any of its descendents is timed-out. Note that this notion of infeasibility need not persist. A goal G may, at some point, be infeasible, because an action in its current plan is timed-out, but G may again become feasible later on, after the agent has revised its state and computed a new plan. Therefore, the only way to ensure that switching to a new top-level goal for planning is admissible (under the focussed profile) is to first check that infeasible goals will *stay* infeasible. This requires an SR. Hence, we can give the following alternative definition of the focussed profile, which is simpler than our earlier definition.

Definition 2 (Focussed profile: trace-based characterisation). *A focussed agent is an agent that, under no circumstances, will generate an operational trace that includes a state with two top-level goals with children.*

This definition is stronger (more restrictive) than our first definition, but as argued earlier, it is operationally equivalent to that definition, because an agent can only be sure that switching goals will not violate the focussed profile after having executed an SR (or after having performed an analogous check).

Possible extensions. Note that, according to our definition, focussed agents do not deal with more than one top-level goal at a time, but may switch between top-level goals in some situations, as exemplified by the following example.

Example 1. Consider the following (portion of a) trace:

$$\ldots, SR(S, \{\}, S', \tau), PI(S', Gs, S'', \tau'), \ldots$$

with the top-level goals of S, S', S'' given by $\{G_1, G_2\}$. Assume that G_1 already has got a plan in S, i.e. the set of items in $Goals(S) \cup Plan(S)$ with ancestor G_1 is not empty. Assume also that G_2 has no plan in S, i.e. the set of items in $Goals(S) \cup Plan(S)$ with ancestor G_2 is empty. Suppose that all items in the plan for G_1 in S are timed-out at τ, and thus S' is such that $Goals(S')$ is the set of all top-level goals in S' and $Plan(S') = \{\}$. Suppose also that neither G_1 nor G_2 are timed-out or achieved at τ', but PI is introducing a plan for G_2, so that the set of items in $Goals(S'') \cup Plan(S'')$ with ancestor G_2 is not empty. The agent with this trace is focussed according to definition 2. However, it does switch from dealing with goal G_1 to dealing with goal G_2, despite goal G_1 being still unachieved and feasible.

[3] The need for this last item will become clear in Example 1.

Definition 2 of focussed agent may be modified to prevent goal switching, by comparing successive agent states in traces and force that once an agent has been planning/executing for one top-most level goal in one state, it must stick to that goal in successive states, until the goal has been achieved or has become unachievable. This would amount to getting rid of the last item in the informal description of focussed agent at the beginning of Section 5.2 (and adding some other suitable conditions instead). This stronger definition of focussed agent would however force extending the notion of cycle theory and operational trace, either by looking at histories of transitions rather than individual transitions when deciding on the next transition, or by introducing additional information into cycle theories, such as variables holding the current top-level goal being dealt with. We therefore leave the stronger definition to future work.
Note also that our notion of focussed agent only refers to *top-level* goals, and not to sub-goals or actions. The notion of focussed agent could be extended so as to define agents that are focussed all the way, from top-level goals down.

Focussed Cycle Theories. To achieve the abstract specification, we need a cycle theory that ensures that before any PI an SR has been performed. This is to ensure that we can proceed with planning for a top-level goal even if some of its current children have become infeasible. However, rather than implementing this behaviour directly, we are going to ensure that PI is only enabled with respect to a set of goals that a focussed agent may plan for given its current state according to the Definition 2. (This, in effect, encourages an SR transition when a PI transition is not enabled.)

Definition 3 (Focussed cycle theories). *A cycle theory is called focussed iff the initial rule $\mathcal{R}_{0|PI}(S, Gs)$ (in $\mathcal{T}_{initial}$) and the basic rule (in $\mathcal{T}_{basic}$) $\mathcal{R}_{T|PI}(S, Gs)$ for any transition T include the enabling condition $focussed(Gs', S, Gs)$, where:*

- *given that Gs is the set of goals to which PI will be applied and $Gs' \supseteq Gs$ is the set of goals returned by the goal selection function, then*
- *the predicate $focussed(Gs', S, Gs)$ holds iff all the goals in Gs are descendants of the same top-level goal (possibly including that top-level goal itself) and no other top-level goal has got any children.*

The focussed variant of the normal cycle theory would have in $\mathcal{T}_{initial}$ the rule

$$\mathcal{R}_{0|PI}(S_0, Gs) : *PI(S_0, Gs) \leftarrow Gs' = c_{GS}(S_0, \tau), \\ focussed(Gs', S_0, Gs), Gs \neq \{\}, now(\tau)$$

instead of the original rule

$$\mathcal{R}_{0|PI}(S_0, Gs) : *PI(S_0, Gs) \leftarrow Gs = c_{GS}(S_0, \tau), Gs \neq \{\}, now(\tau)$$

Similarly, the focussed variant of the normal cycle theory would have in $\mathcal{T}_{basic}$ the rule

$$\mathcal{R}_{AE|PI}(S', Gs) : *PI(S', Gs) \leftarrow AE(S, As, S', \tau'), Gs' = c_{GS}(S', \tau), \\ focussed(Gs', S, Gs), Gs \neq \{\}, now(\tau)$$

instead of the original rule

$$\mathcal{R}_{AE|PI}(S', Gs) : *PI(S', Gs) \leftarrow AE(S, As, S', \tau'), Gs = c_{GS}(S', \tau), Gs \neq \{\}, now(\tau)$$

The *correspondence* between the trace-based characterisation of the *focussed profile* and the class of focussed cycle theories may be stated as follows:

Proposition 2 (Focussed profile). *Any cycle theory that is focussed according to Definition 3 induces the focussed profile of behaviour according to Definition 2.*

Proof. The enabling condition $focussed(Gs', S, Gs)$ restricts the set of goals for which the agent may plan to precisely the set of goals that are available for planning according to the trace-based characterisation of the focussed profile. The claimed correspondence then follows immediately from the fact that PI is the only transition that can add non-top-level goals to a state.

A property of the focussed profile. Let a focussed agent be one equipped with a focussed cycle theory, and a normal agent be one equipped with the normal cycle theory. Then if the two agents have a set of goals for which they have no compatible plans then the focussed agent may be able to achieve at least some of its goals while the normal agent may not be able to achieve any of the goals. The theorem below shows under what conditions the focussed agent is guaranteed to achieve more of its goals compared to the normal agent. Note that conditions 1-6 simply set the scene for the theorem whereas conditions 7-9 restrict features of the environment and the application.

Theorem 2. *Let f be a focussed agent and n be a normal agent. Let f and n be in a state $S = \langle KB, Goals, Plan, TCS \rangle$ at time τ such that all the conditions below hold:*

1. *$Plan$ is empty.*
2. *$Goals$ consists of top-level goals $G_1, \dots, G_n$, $n > 1$* [4].
3. *The goal selection function, in state S, at all times τ', $\tau' \geq \tau$, selects the same set of k goals for some $1 < k \leq n$, until one or more such goals are achieved. Assume these goals are $\{G_1, \dots, G_k\}$, without loss of generality.*
4. *The agents' PI transition produces a total plan for all its input goals.*
5. *At all times after τ, given input goals $\{G_1, \dots, G_k\}$, the agents' PI transition returns no plan, because none exists in the overall state.*
6. *At all times after τ, given input goals $\{G_i\}$, $i = 1, \dots k$, the agents' PI transition returns a (total) plan.*

Then, f will achieve at least one of the goals amongst $G_1, \dots, G_n$, while n will achieve none of them, provided that:

7. *The agents' RE transition generates no goals or actions.*
8. *No POI, AOI transitions are performed, and no GI transition is performed after the establishment of top-level goals $G_1, \dots, G_n$.*
9. *Goals and actions are non-time critical, i.e. no goal or action is timed out.*

Proof. (Sketch) Consider the case of the normal agent n: by conditions 3,5,7,8 the state of n remains the same (although time progresses). In this state, by conditions 3 and 5, n can never make any progress towards achieving any of its top-level goals.
Now consider the case of the focussed agent f: At some time τ_1, $\tau_1 \geq \tau$, f performs PI. By conditions 3 and 6 and the definition of the focussed profile a goal G_i, $i = 1, \dots k$, is selected and PI succeeds in producing a complete plan for G_i, and updates its state by adding all the produced actions As to its $Plan$ and updating TCS appropriately.

[4] Conditions 1. and 2. can arise, for example, if f and n have just executed GI starting from the same initial state.

These new actions will then all be executed. They will not be timed-out by condition 9. So they may be removed from the state of the agent by SR only if their associated goal is achieved. Any new goals and actions that may be introduced by later applications of PI will not interfere with the execution of the actions in As. Therefore, finally, after all the actions are executed, it will be possible to prove by the Temporal Reasoning that goal G_i which was selected at time τ_1 is achieved.

Note that conditions 7–9 are sufficient but not necessary conditions. For example condition 8 can be replaced with one that requires only that any observation recorded as a result of a POI is "independent" of the goals $G_1, \ldots, G_n$, and allows GI transitions but imposes restrictions on their frequency. It is possible to construct examples where some, possibly many, of conditions 7–9 do not hold, but still the focussed agent performs better than the normal one in goal achievement terms.

6 Conclusion

In this paper, building on our earlier work [6], we have further explored the use of cycle theories for declarative control of agents. We showed how in the case of KGP agents we can define concrete and useful agent profiles or personalities by varying the rules in cycle theories. We showed two such profiles in detail, careful and focussed, and exemplified and formally proved their advantages. The cycle theories for these two profiles are no more complicated than the normal cycle theory, and possibly, in the case of the careful profile, the cycle theory is simpler.
The careful profile is best suited to a dynamic unpredictable environment, but one in which the agent does not have strict deadlines. The focussed profile is best suited to resource-bounded agents. The theoretical analysis of the profiles not only allows exploration of heterogeneity of agents, but it can also provide guidelines to designers of agents and implementers, for example those using the PROSOCS platform. There is scope for exploring a number of other profiles, some of which have been introduced in [1]. Exploring other profiles, parameterising their advantages and disadvantages according to factors in the environment and application domains and exploring how profiles can be usefully combined are subjects of current and future research. Currently we see no problem in combining the careful and focussed profiles.
Our work on profiles shares some of the objectives of the work on commitment strategies based on the BDI model [11]. Three commitment strategies have been defined, *blind*, *single minded*, and *open minded*. They are defined by expressing relationships between current and future intentions. A blindly committed agent, for example, maintains its intentions as long as it believes that it has achieved them, while a single minded agent maintains its intentions until it believes they are achievable. Our work on profiles and their consequences goes some way beyond these commitment strategies.
Our approach shares the aims of 3APL [4] to make it possible to program the agent cycle and make the selection mechanisms explicit. But it goes beyond 3APL by abandoning the concept of fixed cycles and replacing it with dynamic programmable cycle theories.

Acknowledgments

This work was partially funded by the IST programme of the EC, FET under the IST-2001-32530 SOCS project, within the GC proactive initiative. We are also grateful to A.C. Kakas and U. Endrich for early discussions regarding this work.

References

1. F. Athienitou, A. Bracciali, U. Endriss, A.C. Kakas, W. Lu, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. Profile related properties. Technical report, SOCS deliverable, 2005.
2. A. Bracciali, N. Demetriou, U. Endriss, A.C. Kakas, W. Lu, P. Mancarella, F. Sadri, K. Stathis, G. Terreni, and F. Toni. The KGP model of agency for global computing: Computational model and prototype implementation. In *Global Computing 2004 Workshop*, page 342. Springer Verlag LNCS 3267, 2005.
3. Y. Dimopoulos and A. C. Kakas. Logic programming without negation as failure. In *Proc. ILPS*, pages 369–384, 1995.
4. K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J. Ch. Meyer. Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.
5. A. C. Kakas, P. Mancarella, and P. M. Dung. The acceptability semantics for logic programs. pages 504–519, 1994.
6. A. C. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. Declarative agent control. In *Proc. CLIMA V*, 2004.
7. A. C. Kakas and P. Moraitis. Argumentation based decision making for autonomous agents. pages 883–890, Melbourne, Victoria, July 14–18 2003.
8. A.C. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. The KGP model of agency. In *Proc. ECAI-2004*, 2004.
9. R.A. Kowalski and F. Toni. Abstract argumentation. *Artificial Intelligence and Law Journal, Special Issue on Logical Models of Argumentation*, 4:275–296, 1996.
10. H. Prakken and G. Sartor. A system for defeasible argumentation, with defeasible priorities. In *International Conference on Formal and Applied Practical Reasoning, Springer Lecture Notes in AI 1085*, pages 510–524. 1996.
11. A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In *Readings in Agents*, pages 317–328. 1997.
12. K. Stathis, A.C. Kakas, W. Lu, N. Demetriou, U. Endriss, and A. Bracciali. PROSOCS: A platform for programming software agents in computational logic. In *Proc. AT2AI*, 2004.

A Normal cycle theory in full

– $\mathcal{T}_{initial}$:
$\mathcal{R}_{0|GI}(S_0, \{\}) : *GI(S_0, \{\}) \leftarrow empty\_goals(S_0)$
$\mathcal{R}_{0|PI}(S_0, Gs) : *PI(S_0, Gs) \leftarrow Gs = c_{GS}(S_0, \tau), Gs \neq \{\}, now(\tau)$
$\mathcal{R}_{0|POI}(S_0, \{\}) : *POI(S_0, \{\}) \leftarrow poi\_pending(\tau), now(\tau)$

– $\mathcal{T}_{basic}$:
rules for deciding what might follow AE:
$\mathcal{R}_{AE|PI}(S', Gs) : *PI(S', Gs) \leftarrow AE(S, As, S', \tau'), Gs = c_{GS}(S', \tau), Gs \neq \{\}, now(\tau)$
$\mathcal{R}_{AE|AE}(S', As') : *AE(S', As') \leftarrow AE(S, As, S', \tau'), As' = c_{AS}(S', \tau), As' \neq \{\}, now(\tau)$
$\mathcal{R}_{AE|AOI}(S', Fs) : *AOI(S', Fs) \leftarrow AE(S, As, S', \tau'), Fs = c_{FS}(S', \tau), Fs \neq \{\}, now(\tau)$
$\mathcal{R}_{AE|SR}(S', \{\}) : *SR(S', \{\}) \leftarrow AE(S, As, S', \tau')$
$\mathcal{R}_{AE|GI}(S', \{\}) : *GI(S', \{\}) \leftarrow AE(S, As, S', \tau')$
rules for deciding what might follow SR:

$\mathcal{R}_{SR|PI}(S', Gs) : *PI(S', Gs) \leftarrow SR(S, \{\}, S', \tau'), Gs = c_{GS}(S', \tau), Gs \neq \{\}, now(\tau)$
$\mathcal{R}_{SR|GI}(S', \{\}) : *GI(S', \{\}) \leftarrow SR(S, \{\}, S'\tau'), Gs = c_{GS}(S', \tau), Gs = \{\}, now(\tau)$
rules for deciding what might follow PI:
$\mathcal{R}_{PI|AE}(S', As) : *AE(S', As) \leftarrow PI(S, Gs, S', \tau'), As = c_{AS}(S', \tau), As \neq \{\}, now(\tau)$
$\mathcal{R}_{PI|SI}(S', Ps) : *SI(S', Ps) \leftarrow PI(S, Gs, S', \tau'), Ps = c_{PS}(S', \tau), Ps \neq \{\}, now(\tau)$
rules for deciding what might follow GI:
$\mathcal{R}_{GI|RE}(S', \{\}) : *RE(S', \{\}) \leftarrow GI(S, \{\}, S', \tau)$
$\mathcal{R}_{GI|PI}(S', Gs) : *PI(S', Gs) \leftarrow GI(S, \{\}, S', \tau'), Gs = c_{GS}(S', \tau), Gs \neq \{\}, now(\tau)$
rules for deciding what might follow RE:
$\mathcal{R}_{RE|PI}(S', Gs) : *PI(S', Gs) \leftarrow RE(S, \{\}, S', \tau'), Gs = c_{GS}(S', \tau), Gs \neq \{\}, now(\tau)$
$\mathcal{R}_{RE|SI}(S', Ps) : *SI(S', Ps) \leftarrow RE(S, \{\}, S', \tau'), Ps = c_{PS}(S', \tau), Ps \neq \{\}, now(\tau)$
$\mathcal{R}_{RE|AE}(S', As) : *AE(S', As) \leftarrow RE(S, \{\}, S', \tau'), As = c_{AS}(S', \tau), As \neq \{\}, not(\tau)$
$\mathcal{R}_{RE|SR}(S', \{\}) : *SR(S', \{\}) \leftarrow RE(S, \{\}, S', \tau')$
rules for deciding what might follow SI:
$\mathcal{R}_{SI|AE}(S', As) : *AE(S', As) \leftarrow SI(S, Ps, S', \tau'), As = c_{AS}(S', \tau), As \neq \{\}, now(\tau)$
rules for deciding what might follow AOI:
$\mathcal{R}_{AOI|AE}(S', As) : *AE(S', As) \leftarrow AOI(S, Fs, S', \tau'), As = c_{AS}(S', \tau), As \neq \{\}, now(\tau)$
$\mathcal{R}_{AOI|SR}(S', \{\}) : *SR(S', \{\}) \leftarrow AOI(S, Fs, S', \tau')$
$\mathcal{R}_{AOI|SI}(S', Ps) : *SI(S', Ps) \leftarrow AOI(S, Fs, S', \tau')Ps = c_{PS}(S', \tau), Ps \neq \{\}, now(\tau)$
rules for deciding when POI should take place:
$\mathcal{R}_{T|POI}(S', \{\}) : *POI(S', \{\}) \leftarrow T(S, X, S', \tau'), poi\_pending(\tau), now(\tau)$
for all transitions T;
rules for deciding what might follow POI:
$\mathcal{R}_{POI|GI}(S', \{\}) : *GI(S', \{\}) \leftarrow POI(S, \{\}, S', \tau)$
$\mathcal{R}_{POI|RE}(S', \{\}) : *RE(S', \{\}) \leftarrow POI(S, \{\}, S', \tau)$
$\mathcal{R}_{POI|SR}(S', \{\}) : *SR(S', \{\}) \leftarrow POI(S, \{\}, S', \tau)$

– $\mathcal{T}_{behaviour}$:
GI is given higher priority if there are no goals in $Goals$ and actions in $Plan$:
$\mathcal{P}^{T}_{GI \succ T'} : \mathcal{R}_{T|GI}(S, \{\}) \succ \mathcal{R}_{T|T'}(S, X) \leftarrow empty\_goals(S), empty\_plan(S)$
for all T, T', with $T' \neq GI$ and T possibly 0;
GI is also given higher priority after a POI:
$\mathcal{P}^{POI}_{GI \succ T} : \mathcal{R}_{POI|GI}(S') \succ \mathcal{R}_{POI|T}(S, S')$ for all $T \neq GI$;
after GI, RE is given higher priority:
$\mathcal{P}^{GI}_{RE \succ T} : \mathcal{R}_{GI|RE}(S, \{\}) \succ \mathcal{R}_{GI|T}(S, X)$ for all $T \neq RE$;
after RE, PI is given higher priority:
$\mathcal{P}^{RE}_{PI \succ T} : \mathcal{R}_{RE|PI}(S, Gs) \succ \mathcal{R}_{RE|T}(S, X)$ for all $T \neq PI$;

after PI, AE is given higher priority, unless there are actions in the actions selected for execution whose preconditions are "unreliable" and need checking, in which case SI will be given higher priority:
$\mathcal{P}^{PI}_{AE \succ T} : \mathcal{R}_{PI|AE}(S, As) \succ \mathcal{R}_{PI|T}(S, X) \leftarrow not\, unreliable\_pre(As)$
for all $T \neq AE$;
$\mathcal{P}^{PI}_{SI \succ T} : \mathcal{R}_{PI|SI}(S, Ps) \succ \mathcal{R}_{PI|T}(S, As) \leftarrow unreliable\_pre(As)$
for all $T \neq SI$;
after SI, AE is given higher priority:
$\mathcal{P}^{SI}_{AE \succ T} : \mathcal{R}_{SI|AE}(S, As) \succ \mathcal{R}_{SI|T}(S, X)$ for all $T \neq AE$;
after AE, AE should be given higher priority until there are no more actions to execute in $Plan$, in which case either AOI or SR should be given higher priority, depending on whether there are actions which are "unreliable", in the sense that their effects need checking, or not:
$\mathcal{P}^{AE}_{AE \succ T} : \mathcal{R}_{AE|AE}(S, As) \succ \mathcal{R}_{AE|T}(S, X)$ for all $T \neq AE$;
$\mathcal{P}^{AE}_{AOI \succ T} : \mathcal{R}_{AE|AOI}(S, Fs) \succ \mathcal{R}_{AE|T}(S, X) \leftarrow empty\_executable\_plan(S), unreliable\_post(S)$
for all $T \neq AOI$;
$\mathcal{P}^{AE}_{SR \succ T} : \mathcal{R}_{AE|SR}(S, \{\}) \succ \mathcal{R}_{AE|T}(S, X) \leftarrow empty\_executable\_plan(S), not\, unreliable\_post(S)$
for all $T \neq SR$;
after SR, PI should have higher priority:
$\mathcal{P}^{SR}_{PI \succ T} : \mathcal{R}_{SR|PI}(S, Gs) \succ \mathcal{R}_{SR|T}(S, \{\})$ for all $T \neq PI$;
after any transition, POI is preferred over all other transitions:
$\mathcal{P}^{T}_{PI \succ T'} : \mathcal{R}_{T||OI}(S) \succ \mathcal{R}_{T|T'}(S, X)$ for all T, T', with $T' \neq POI$ and T possibly 0;
in the initial state, PI is given higher priority:
$\mathcal{P}^{0}_{PI \succ T} : \mathcal{R}_{0|PI}(S, Gs) \succ \mathcal{R}_{0|T}(S, X)$ for all $T \neq PI$;

– The auxiliary part includes definitions for $empty\_goals$, $unreliable\_pre$, $unreliable\_post$, $empty\_executable\_plan$, $poi\_pending$ etc. Note that $poi\_pending(\tau)$ holds when there is an input from the environment pending.

Diagnosis of plan execution and the executing agent[*]

Nico Roos[1] and Cees Witteveen[2]

[1] IKAT, Universiteit Maastricht
P.O.Box 616, NL-6200 MD Maastricht
roos@cs.unimaas.nl

[2] Faculty EEMCS, Delft University of Technology
P.O.Box 5031, NL-2600 GA Delft
witt@ewi.tudelft.nl

Abstract. We discuss the application of Model-Based Diagnosis in (agent-based) planning. Here, a plan together with its executing agent is considered as a system to be diagnosed. It is assumed that the execution of a plan can be monitored by making partial observations of the results of actions. These observations are used to explain the observed deviations from the plan by qualifying some action instances that occur in the plan as behaving abnormally. Unlike in standard model-based diagnosis, however, in plan diagnosis we cannot assume that actions fail independently. We focus on two sources of dependencies between failures: such failings may occur as the result of malfunctioning of the executing agent or may be caused by dependencies between action instances occurring in a plan. Therefore, we introduce causal rules that relate health states of the agent and health states of actions to abnormalities of other action instances. These rules enable us to determine the underlying causes of plan failing and to predict future anomalies in the execution of actions.

1 Introduction

The well-known quote: *"No plan survives its first contact with the enemy"* should remind us that *diagnosis* constitutes an unavoidable part of the plan execution process.[3] Since there is a huge number of potential factors that might influence, or even prevent, correct plan execution, it is not surprising that current approaches to plan diagnosis are rather diverse.

The aim of this paper is to adapt and extend a classical Model-Based Diagnosis (MBD) approach to the diagnosis of plans. To this end, first we will show how a plan consisting of a partially ordered set of actions can be viewed as a system to be diagnosed and how a diagnosis can be established using *partial observations* of a plan in progress. Distinguishing between normal and abnormal execution of actions in a plan, we then introduce sets of actions qualified as abnormal to explain the deviations

[*] This research is supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Ministry of Economic Affairs (the Netherlands). Project DIT5780: Distributed Model Based Diagnosis and Repair

[3] The quote is attributed to the Prussian Field Marshall Von Moltke.

between expected plan states and observed plan states. Hence, in this approach, a plan diagnosis is just a set of abnormal actions that is able to explain the deviations observed. Although plan diagnosis conceived in this way is a rather straightforward application of MBD to plans, we do need to introduce new criteria for selecting acceptable plan diagnoses: First of all, while in standard MBD usually subset-minimal diagnoses, or within them *minimum (cardinality)* diagnoses, are preferred, we also prefer *maximum informative* diagnoses. The latter type of diagnosis maximizes the exact similarity between predicted and observed plan states. Although maximum informative diagnoses are always subset minimal, they are not necessarily of minimum cardinality. More differences between MBD and plan diagnosis appear if we take a detailed look into the reasons for choosing minimal diagnoses. The idea of establishing a minimal diagnosis in MBD is governed by the principle of *minimal change*: explain the abnormalities in the behavior observed by changing the qualification from normal to abnormal for as few system components as necessary. Using this principle is intuitively acceptable if the components qualified as abnormal are failing *independently*. However, as soon as *dependencies* exist between such components, the choice for minimal diagnoses cannot be justified. As we will argue, the existence of dependencies between failing actions in a plan is often the rule instead of an exception. Therefore, we will refine the concept of a plan diagnosis by introducing the concept of a *causal diagnosis*. To establish such a causal diagnosis, we consider both the executing agent and its plan as constituting the system to be diagnosed and we explicitly relate health states of the executing agent and subsets of (abnormally qualified) actions to the abnormality of other actions in the form of causal rules. These rules enable us to replace a set of dependent failing actions (e.g. a plan diagnosis) by a set of unrelated *causes* of the original diagnosis. This independent and usually smaller set of causes constitutes a causal diagnosis, consisting of a health state of an agent and an independent (possibly empty) set of failing actions. Such a causal diagnosis always generates a cover of a minimal diagnosis. More importantly, such causal diagnoses can also be used to predict failings of actions that have to be executed in the plan and thereby also can be used to assess the consequences of such failures for goal realizability.

This paper is organized as follows. First of all, in the next section, we place our approach into perspective by discussing some related approaches to plan diagnosis. Section 3 introduces the preliminaries of plan-based diagnosis, while Section 4 formalizes plan-based diagnosis. Section 5 extends the formalization to determining the agent's health state. Finally, we briefly discuss some computational aspects of (causal) plan diagnosis.

2 Related research

In this section we briefly discuss some other approaches to plan diagnosis. Like we use MBD as a starting point to plan diagnosis, Birnbaum et al. [1] apply MBD to *planning agents* relating health states of agents to *outcomes* of their planning activities, but not taking into account faults that can be attributed to actions occurring in a plan as a separate source of errors. However, instead of focusing upon the relationship between agent properties and outcomes of plan executions, we take a more detailed approach, distinguishing two separate sources of errors (actions and properties of the executing agents) and focusing upon the detection of anomalies during the plan execution. This enables us to predict the outcomes of a plan on beforehand instead of using them only as observations.

Another approach that directly applies model-based diagnosis to plan execution has been proposed in [6]. Here, the authors focus on agents each having an individual plan, and where conflicts between these plans may arise (e.g. if they require the same resource). Diagnosis is applied to determine those factors that are accountable for *future* conflicts. The authors, however, do not take into account dependencies between health modes of actions and do not consider agents that collaborate to execute a common plan.

Kalech and Kaminka [10, 11] apply *social diagnosis* in order to find the cause of an anomalous plan execution. They consider hierarchical plans consisting of so-called *behaviors*. Such plans do not prescribe a (partial) execution order on a set of actions. Instead, based on its observations and beliefs, each agent chooses the appropriate behavior to be executed. Each behavior in turn may consist of primitive actions to be executed, or of a set of other behaviors to choose from. Social diagnosis then addresses the issue of determining what went wrong in the joint execution of such a plan by identifying the disagreeing agents and the causes for their selection of incompatible behaviors (e.g., belief disagreement, communication errors). This approach might complement our approach when conflicts not only arise as the consequence of faulty actions, but also as the consequence of different selections of sub-plans in a joint plan.

Lesser et al. [3, 9] also apply diagnosis to (multi-agent) plans. Their research concentrates on the use of a *causal model* that can help an agent to refine its initial diagnosis of a failing *component* (called a *task*) of a plan. As a consequence of using such a causal model, the agent would be able to generate a new, situation-specific plan that is better suited to pursue its goal. While their approach in its ultimate intentions (establishing anomalies in order to find a suitable plan repair) comes close to our approach, their approach to diagnosis concentrates on specifying the exact causes of the failing of one single *component* (task) of a plan. Diagnosis is based on observations of a component without taking into account the consequences of failures of such a component w.r.t. the remaining plan. In our approach, instead, we are interested in applying MBD-inspired methods to *detect* plan failures. Such failures are based on observations during plan execution and may concern individual components of the plan, but also agent properties. Furthermore, we do not only concentrate on failing components themselves, but also on the consequences of these failures for the future execution of plan elements.

3 Preliminaries

3.1 Model based Diagnosis

In Model-Based Diagnosis (MBD) [4, 5, 13] a system S is modeled as consisting of a set $Comp$ of components and their relations, for each component $c \in Comp$ a set H_c of *health modes* is distinguished and for each health mode $h_c \in H_c$ of each component c a specific (input-output) behavior of c is specified. Given some input to S, its output is defined if the health mode of each component $c \in Comp$ is known. The diagnostic engine is triggered whenever, under the assumption that all components are functioning normally, there is a discrepancy between the output as predicted from the input observations, and the actually observed output. The result of MBD is a suitable assignment of health modes to the components, called a *diagnosis*, such that the actually observed output is *consistent* with this health mode qualification or can be *explained* by this qualification. Usually, in a diagnosis one requires the number of components qualified as abnormally to be minimized.

3.2 States

We consider plan-based diagnosis as a simple extension of model-based diagnosis where the model is not a description of an underlying system but a *plan* of an agent. Before we discuss plans, we discuss our *object*- or *resource-based* view on the world, assuming that for the planning problem at hand, the world can be simply described by a set $Obj = \{o_1, o_2, \ldots, o_n\}$ of objects, their respective *value domains* S_i and their (current) values $s_i \in S_i$.[4] A *state of the world* σ then is an element of $S_1 \times S_2 \times \ldots \times S_n$. It will not always be possible to give a complete state description. Therefore, we introduce a *partial state* as an element $\pi \in S_{i_1} \times S_{i_2} \times \ldots \times S_{i_k}$, where $1 \leq k \leq n$ and $1 \leq i_1 < \ldots < i_k \leq n$. We use $O(\pi)$ to denote the set of objects $\{o_{i_1}, o_{i_2}, \ldots, o_{i_k}\} \subseteq Obj$ specified in such a state π. The value s_j of object $o_j \in O(\pi)$ in π will be denoted by $\pi(j)$. The value of an object $o_j \in Obj$ not occurring in a partial state π is said to be unknown (or unpredictable) in π, denoted by $\bot$. Partial states can be ordered with respect to their information content: π is said to be contained in π', denoted by $\pi \sqsubseteq \pi'$, iff $O(\pi) \subseteq O(\pi')$ and $\pi'(j) = \pi(j)$ for every $o_j \in O(\pi)$. We say that two partial states π, π' are *equivalent* modulo a set of objects O, denoted by $\pi =_O \pi'$, if for every $o_j \in O$, $\pi(j) = \pi'(j)$. Finally, we define the partial state π restricted to a given set O, denoted by $\pi \upharpoonright O$, as the state $\pi' \sqsubseteq \pi$ such that $O(\pi') = O \cap O(\pi)$.

3.3 Goals

An (elementary) goal g of an agent specifies a set of states an agent wants to bring about using a plan. Here, we specify each such a goal g as a constraint, that is a relation over some product $S_{i_1} \times \ldots \times S_{i_k}$ of domains.

We say that a goal g is satisfied by a partial state π, denoted by $\pi \models g$, if the relation g contains at least one tuple $(v_{i_1}, v_{i_2}, \ldots, v_{i_k})$ such that $(v_{i_1}, v_{i_2}, \ldots v_{i_k}) \sqsubseteq \pi$. We assume each agent to have a set G of such elementary goals $g \in G$. We use $\pi \models G$ to denote that all goals in G hold in π, i.e. for all $g \in G$, $\pi \models g$.

3.4 Actions and action schemes

An *action scheme* or plan operator α is represented as a function that replaces the values of a subset $O_\alpha \subseteq Obj$ by other values, dependent upon the values of another set $O'_\alpha \supseteq O_\alpha$ of objects. Hence, every action scheme α can be modeled as a (partial) function $f_\alpha : S_{i_1} \times \ldots \times S_{i_k} \to S_{j_1} \times \ldots \times S_{j_l}$, where $1 \leq i_1 < \ldots < i_k \leq n$ and

[4] In contrast to the conventional approach to state-based planning, cf. [8].

$\{j_1, \ldots, j_l\} \subseteq \{i_1, \ldots, i_k\}$. The objects whose value domains occur in $dom(f_\alpha)$ will be denoted by $dom_O(\alpha) = \{o_{i_1}, \ldots, o_{i_k}\}$ and, likewise $ran_O(\alpha) = \{o_{j_1}, \ldots, o_{j_l}\}$. Note that it is required that $ran_O(\alpha) \subseteq dom_O(\alpha)$. This functional specification f_α constitutes the *normal* behavior of the action scheme, denoted by f_α^{nor}.

Example 1. Figure 1 depicts two states σ_0 and σ_1 (the white boxes) each characterized by the values of four objects o_1, o_2, o_3 and o_4. The partial states π_0 and π_1 (the gray boxes) characterize a subset of values in a (complete) state. Action schemes are used to model state changes. The domain of the action scheme α is the subset $\{o_1, o_2\}$, which are denoted by the arrows pointing to α. The range of α is the subset $\{o_1\}$, which is denoted by the arrow pointing from α. Finally, the dashed arrow denotes that the value of object o_2 is not changed by operator(s) causing the state change. ∎

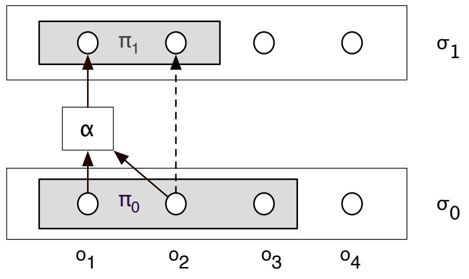


Fig. 1. Plan operators & states.

The correct execution of an action may fail either because of an inherent malfunctioning or because of a malfunctioning of an agent responsible for executing the action, or because of unknown external circumstances. In all these cases we would like to model the effects of executing such failed actions. Therefore, we introduce a set of *health modes* M_α for each action scheme α. This set M_α contains at least the normal mode nor, the mode ab indicating the most general abnormal behavior, and possibly several other specific fault modes. The most general abnormal behavior of action α is specified by the function f_α^{ab}, where $f_\alpha^{ab}(s_{i_1}, s_{i_2}, \ldots, s_{i_k}) = (\perp, \perp, \ldots, \perp)$ for every partial state $(s_{i_1}, s_{i_2}, \ldots, s_{i_k}) \in dom(f_\alpha)$.[5] To keep the discussion simple, in the sequel we distinguish only the health modes nor and ab.

Given a set $\mathcal{A}$ of action schemes, we will need to consider a set $A \subseteq inst(\mathcal{A})$ of *instances* of actions in $\mathcal{A}$. Such instances will be denoted by small roman letters a_i. If $type(a_i) = \alpha \in \mathcal{A}$, such an instance a_i is said to be of *type* α. If the context permits we will use "actions" and "instances of actions" interchangeably.

[5] This definition implies that the behavior of abnormal actions is essentially unpredictable.

3.5 Plans

A plan is a tuple $P = \langle \mathcal{A}, A, < \rangle$ where $A \subseteq Inst(\mathcal{A})$ is a set of instances of actions occurring in $\mathcal{A}$ and $(A, <)$ is a partial order. The partial order relation $<$ specifies a precedence relation between these instances: $a < a'$ implies that the instance a must finish before the instance a' may start. We will denote the *transitive reduction* of $<$ by $\ll$, i.e., $\ll$ is the smallest subrelation of $<$ such that the transitive closure $\ll^+$ of $\ll$ equals $<$.

We assume that if in a plan P two action instances a and a' are independent, in principle they may be executed concurrently. This means that the dependency relation $<$ at least should capture all resource dependencies that would prohibit concurrent execution of actions. Therefore, we assume $<$ to satisfy the following *concurrency requirement*:

If $ran_O(a) \cap dom_O(a') \neq \varnothing$ then $a < a'$ or $a' < a$.[6]

That is, for concurrent instances, domains and ranges do not overlap.

Example 2. Figure 2 gives an illustration of a plan. Arrows relate the objects an action uses as inputs and the objects it produces as its outputs to the action itself. In this plan, the dependency relation is specified as $a_1 \ll a_3$, $a_2 \ll a_4$, $a_4 \ll a_5$, $a_4 \ll a_6$ and $a_1 \ll a_5$. Note that the last dependency has to be included because a_5 changes the value of o_2 needed by a_1. The action a_1 shows that not every object occurring in the domain of an action need to be affected by the action. The actions a_5 and a_6 illustrate that concurrent actions may have overlapping domains. ■

4 Standard Plan Diagnosis

Let us assume, for the moment, that each action instance can be viewed as an independent component of a plan. To each action instance a a health mode $m_a \in \{nor, ab\}$ can be assigned and the result is called a *qualified* plan. In establishing which part of the plan fails, we are only interested in those actions qualifies as abnormal. Therefore, we define a qualified version P_Q of a plan $P = \langle \mathcal{A}, A, < \rangle$ as a tuple $P_Q = \langle \mathcal{A}, A, <, Q \rangle$, where $Q \subseteq A$ is the subset of instances of actions qualified as abnormal (and therefore, $A - Q$ the subset of actions qualified as normal).

Since a qualification Q corresponds to assigning the health mode ab to every action in Q and since $f_a^{ab}(s_{i_1}, s_{i_2}, \ldots, s_{i_k}) = (\bot, \bot, \ldots, \bot)$ for every action $a \in Q$ with $type(a) = \alpha$, the results of anomalously executed actions are unpredictable. Note that a "normal" plan P corresponds to the qualified plan $P_\varnothing$ and furthermore that in our context "undefined" is considered to be equivalent to "unpredictable".

4.1 Qualified Plan execution

For simplicity, when a plan P is executed, we will assume that every action takes a unit of time to execute. We are allowed to observe the execution of a plan P at discrete times

[6] Note that since $ran_O(a) \subseteq dom_O(a)$, this requirement excludes overlapping ranges of concurrent actions, but domains of concurrent actions are allowed to overlap as long as the values of the object in the overlapping domains are not affected by the actions.

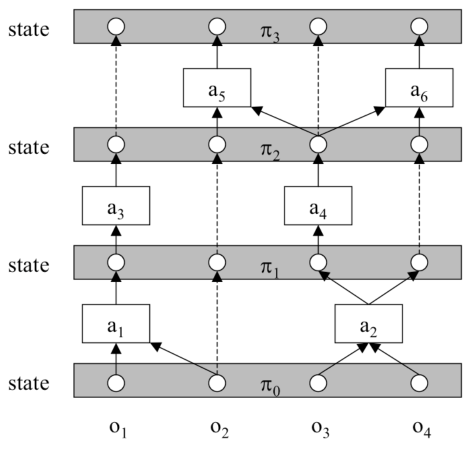


Fig. 2. Plans and action instances. Each state characterizes the values of four objects o_1, o_2, o_3 and o_4. States are changed by application of action instances

$t = 0, 1, 2, \ldots, k$ where k is the depth of the plan, i.e., the longest $<$-chain of actions occurring in P. Let $depth_P(a)$ be the depth of action a in plan $P = \langle \mathcal{A}, A, < \rangle$.[7] We assume that the plan starts to be executed at time $t = 0$ and that concurrency is fully exploited, i.e., if $depth_P(a) = k$, then execution of a has been completed at time $t = k + 1$. Thus, all actions a with $depth_P(a) = 0$ are completed at time $t = 1$ and every action a with $depth_P(a) = k$ will be started at time k and will be completed at time $k+1$. Note that thanks to the above specified concurrency requirement, concurrent execution of actions having the same depth leads to a well-defined result.

Let P_t denote the set of actions a with $depth_P(a) = t$, let $P_{>t} = \bigcup_{t'>t} P_{t'}$, $P_{<t} = \bigcup_{t'<t} P_{t'}$ and $P_{[t,t']} = \bigcup_{k=t}^{t'} P_k$. Execution of P on a given initial state σ_0 will induce a sequence of states $\sigma_0, \sigma_1, \ldots, \sigma_k$, where σ_{t+1} is generated from σ_t by applying the set of actions P_t to σ_t. Instead, however, of assuming total states and total state transitions, we define the (predicted) effect of the execution of plan P on a given (partial) state π at time $t \geq 0$, denoted by (π, t).

We say that $(\pi', t + 1)$ is (directly) generated by execution of P_Q from (π, t), abbreviated by $(\pi, t) \rightarrow_{Q;P} (\pi', t + 1)$, iff the following conditions hold:

1. $\pi' \restriction ran_O(a) = f_a^{nor}(\pi \restriction dom_O(a))$ for each $a \in P_t - Q$ such that $dom_O(a) \subseteq O(\pi)$, that is, the consequences of all actions a enabled in π can be predicted and occur in π'.[8]

[7] Here, $depth_P(a) = 0$ if $\{a' \mid a' \ll a\} = \varnothing$ and $depth_P(a) = 1 + max\{depth_P(a') \mid a' \ll a\}$, else. If the context is clear, we often will omit the subscript P.

[8] An action a is enabled in a state π if $dom_O(a) \subseteq O(\pi)$.

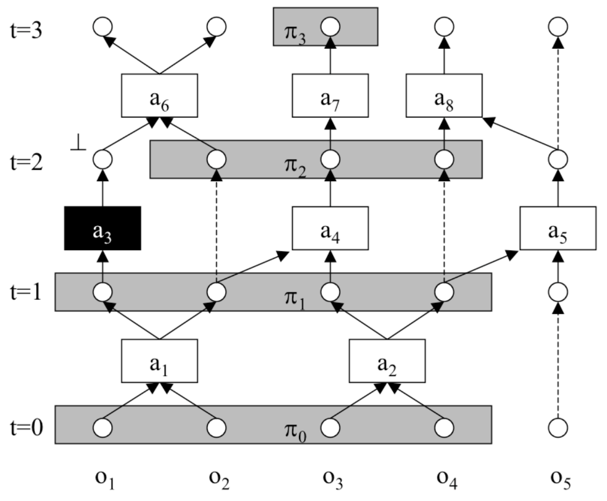


Fig. 3. Plan execution with abnormal actions

2. $O(\pi') \cap ran_O(a) = \varnothing$ for each $a \in Q \cap P_t$, since the result of executing an abnormal action cannot be predicted (even if such an action is enabled in π);
3. $O(\pi') \cap ran_O(a) = \varnothing$ for each $a \in P_t$ with $dom_O(a) \not\subseteq O(\pi)$, that is, even if an action a is enabled in (the complete state) σ_t, if a is not enabled in $\pi \sqsubseteq \sigma_t$, the result is not predictable and therefore does not occur in π', since it is not possible to predict the consequences of actions that depend on values not defined in π.
4. $\pi'(i) = \pi(i)$ for each $o_i \notin ran_O(P_t)$, that is, the value of any object not occurring in the range of an action in P_t should remain unchanged. Here, $ran_O(P_t)$ is a shorthand for the union of the sets $ran_O(a)$ with $a \in P_t$.

For arbitrary values of $t \leq t'$ we say that (π', t') is (directly or indirectly) generated by execution of P_Q from (π, t), denoted by $(\pi, t) \rightarrow^*_{Q;P} (\pi', t')$, iff the following conditions hold:

1. if $t = t'$ then $\pi' = \pi$;
2. if $t' = t + 1$ then $(\pi, t) \rightarrow_{Q;P} (\pi', t')$;
3. if $t' > t + 1$ then there must exists some state $(\pi'', t' - 1)$ such that $(\pi, t) \rightarrow^*_{Q;P} (\pi'', t' - 1)$ and $(\pi'', t' - 1) \rightarrow_{Q;P} (\pi', t')$.

Note that $(\pi, t) \rightarrow^*_{\varnothing;P} (\pi', t')$ denotes the normal execution of a normal plan $P_\varnothing$. Such a normal plan execution will also be denoted by $(\pi, t) \rightarrow^*_P (\pi', t')$.

Example 3. Figure 3 gives an illustration of an execution of a plan with abnormal actions. Suppose action a_3 is abnormal and generates a result that is unpredictable ($\perp$). Given the qualification $Q = \{a_3\}$ and the partially observed state π_0 at time point $t = 0$, we predict the partial states π_i as indicated in Figure 3, where $(\pi_0, t_0) \rightarrow^*_{Q;P} (\pi_i, t_i)$ for $i = 1, 2, 3$. Note that since the value of o_1 and of o_5 cannot be predicted at time

$t = 2$, the result of action a_6 and of action a_8 cannot be predicted and π_3 contains only the value of o_3. ■

4.2 Diagnosis

Suppose now that we have a (partial) observation $obs(t) = (\pi, t)$ of the state of the world at time t and an observation $obs(t') = (\pi', t')$ at time $t' > t \geq 0$ during the execution of the plan P. We would like to use these observations to infer the health states of the actions occurring in P. Assuming a normal execution of P, we can (partially) predict the state of the world at a time point t' given the observation $obs(t)$: if all actions behave normally, we predict a partial state $\pi'_\varnothing$ at time t' such that $obs(t) \rightarrow^*_P (\pi'_\varnothing, t')$. Since we do not require observations to be made systematically, $O(\pi')$ and $O(\pi'_\varnothing)$ might only partially overlap. Therefore, if this assumption holds, the values of the objects that occur in both the predicted state and the observed state at time t' should match, i.e, we should have

$$\pi' =_{O(\pi') \cap O(\pi'_\varnothing)} \pi'_\varnothing.$$

If this is not the case, the execution of some action instances must have gone wrong and we have to determine a qualification Q such that the predicted state derived using Q agrees with π'. This is nothing else then a straight-forward extension of the diagnosis concept in MBD to plan diagnosis (cf. [5]):

Definition 1. *Let* $P = \langle \mathcal{A}, A, < \rangle$*be a plan with observations* $obs(t) = (\pi, t)$ *and* $obs(t') = (\pi', t')$*, where* $t < t' \leq depth(P)$ *and let* $obs(t) \rightarrow^*_{Q;P} (\pi'_Q, t')$ *be a derivation assuming a qualification* Q*. Then* Q *is said to be a* plan diagnosis *of* $\langle P, obs(t), obs(t') \rangle$ *iff* $\pi' =_{O(\pi') \cap O(\pi'_Q)} \pi'_Q$.

So in a plan diagnosis Q the observed partial state (π') at time t' and the predicted state (π'_Q) assuming the qualification Q at time t' agree upon the values of all objects occurring in both states.

Example 4. Consider again Figure 3 and suppose that we did not know that action a_3 was abnormal and that we observed $obs(0) = ((s_1, s_2, s_3, s_4), 0)$ and $obs(3) = (s'_1, s'_3, s'_5), 3)$. Using the normal plan derivation relation starting with $obs(0)$ we will predict a state $\pi'_\varnothing$ at time $t = 3$ where $\pi'_\varnothing = (s''_1, s''_2, s''_3)$. If everything is ok, the values of the objects predicted as well as observed at time $t = 3$ should correspond, i.e. we should have $s'_j = s''_j$ for $j = 1, 3$. If, for example, only s'_1 would differ from s''_1, then we could qualify a_6 as abnormal, since then the predicted state at time $t = 3$ using $Q = \{a_6\}$ would be $\pi'_Q = (s''_3)$ and this partial state agrees with the predicted state on the value of o_3. ■

Note that for all objects in $O(\pi') \cap O(\pi'_Q)$, the qualification Q provides an *explanation* for the observation π' made at time point t'. Hence, for these objects the qualification provides an *abductive diagnosis* [4] for the normal observations. For all observed objects in $O(\pi') - O(\pi'_Q)$, no value can be predicted given the qualification Q. Hence, by declaring them to be unpredictable, possible conflicts with respect to these objects if a normal execution of all actions is assumed, are resolved. This corresponds with the idea of a *consistency-based diagnosis* [13].

If Q is a plan diagnosis of $\langle P, obs(t), obs(t')\rangle$, then every superset $Q' \supseteq Q$ is also a plan diagnosis, since in that case we have $\pi'_{Q'} \sqsubseteq \pi'_Q$ and therefore $\pi' =_{O(\pi')\cap O(\pi'_Q)} \pi'_Q$ implies $\pi' =_{O(\pi')\cap O(\pi'_{Q'})} \pi'_{Q'}$. Clearly then, the smaller a diagnosis is, the more values it will predict that are also actually observed in the resulting plan state. This, like in MBD, is a reason for us to prefer *minimum* diagnoses among the set of minimal diagnoses.

But there is a caveat: a minimum diagnosis only minimizes abnormalities to explain deviations; as important however for a diagnosis might be its *information content*, i.e. the exactness it provides in predicting the values of the variables occurring in the observed state π'. This means that besides *minimizing* the cardinality of abnormalities another criterion could be *maximizing* the exactness of the similarity by maximizing $|O(\pi') \cap O(\pi'_Q)|$ i.e. maximizing the number of variables having the same value in the predicted state and the observed state. Therefore, besides a minimum diagnosis we also define the notion of a *maximum informative diagnosis*:

Definition 2. *Given plan observations $\langle P, (\pi, t), (\pi', t')\rangle$, a qualification Q is said to be a* minimum plan diagnosis *if for every plan diagnosis Q' it holds that $|Q| \leq |Q'|$.*

Q is said to be a maximum informative plan-diagnosis *iff for all plan diagnoses Q^*, it holds that $|O(\pi') \cap O(\pi'_Q)| \geq |O(\pi') \cap O(\pi'_{Q^*})|$.*

Note that for every maximum informative diagnosis Q we have $O(\pi') \cap O(\pi'_Q) \subseteq O(\pi') \cap O(\pi'_\varnothing)$, where $obs(t) \rightarrow^*_{\varnothing;P} (\pi'_\varnothing, t')$ is the partial state derivation assuming a *normal plan* execution.

Also note that every maximum informative diagnosis is a minimal diagnosis. So both minimum plan diagnoses and maximum informative plan diagnoses are the result of different criteria for selecting minimal diagnoses, as the following example shows:

Example 5. To illustrate the difference between minimum plan diagnosis en maximum informative diagnosis, consider again the plan execution depicted in Figure 3. Given $obs(0)$ and $obs(3)$ and a deviation in the value of o_2 at time $t = 3$, there are three possible minimum diagnoses: $D_1 = \{a_1\}$, $D_2 = \{a_3\}$ and $D_3 = \{a_6\}$. D_2 and D_3 are also maximum-informative diagnoses. ■

5 Causes of plan-execution failures

Unlike in classical MBD, minimum diagnosis and maximum-informative diagnosis need not provide the best explanation for the differences between observed effects of a plan execution and the predicted effects. The reason is that often in a plan instances of actions do not fail independently. For example, suppose that we have a plan for carrying luggage from a depot to a number of waiting planes. Such a plan might contain several instances of a drive action pertaining to the same carrier controlled by an agent. Suppose that an instance a_i of some drive action (type) α behaves abnormally because of malfunctioning of the carrier. Then it is reasonable to assume that other instances a_j of the same drive action that occur in the plan *after* a_i can be predicted to behave abnormally, too. Another possibility is that a number of instances of actions is related the malfunctioning of an *agent* executing several actions in the plan. For example, in the luggage example, the carrier is controlled is by a driving agent. If this agent itself is not functioning well, all driving actions as well as loading and unloading actions might be affected.

Such dependencies between action instances and between agent health states and action instances imply that sometimes qualifying an instance of an action as being abnormal implies that other instances of actions must be qualified a being abnormal, too. Minimum and information-maximum diagnosis do not take into account these dependencies between action failures. Therefore, we must take into consideration the underlying *causes* of a plan-execution failure.

5.1 Causal Rules

To be able to include a malfunctioning of an executing agent as a possible cause, we will consider a plan together with its executing agent as the system to be diagnosed. Here, an agent will be simply represented by a set H of specific health states. To identify causes of action failures, we use a set R of *causal rules* in combination with plan diagnosis. A causal rule is a rule that can appear in the following forms:

- $(\alpha_1, \alpha_2, \ldots, \alpha_k) \to \alpha_{k+1}$, where $k \geq 1$ and, for $i = 1, 2 \ldots, k+1$, $\alpha_i \in \mathcal{A}$ are action types. This type of rule relates the occurrence of a set of failed actions to the occurrence of a failed action implied by them. The intuitive meaning of these rules is that if during plan execution there are, for $i = 1, \ldots, k$, action instances a_i of type α_i that have been qualified as abnormal up to time t, then it is inferred that from time $t+1$ on all instances of actions of type α_{k+1} will behave abnormally, too.
- $(h; \alpha_1, \alpha_2, \ldots, \alpha_k) \to \alpha_{k+1}$, where $k \geq 0$, $h \in H$ is a health state $(h \neq nor)$ of the plan executing agent and, for $i = 1, 2 \ldots, k+1$, $\alpha_i \in \mathcal{A}$ are action types. This type of rule relates the occurrence of an agent abnormality h and a set of action abnormalities occurring at time t to the inference of a failed action at time $t+1$. The intuitive meaning of such a rule is that if during plan execution at some time $t' \leq t+1$ the agent operates in some abnormal health states h and, for $i = 1, 2, \ldots, k$, there are action instances a_i of type α_i that have been qualified as abnormal up to time t, then it is inferred that from time $t+1$ on all instances of actions of type α_{k+1} that occur in the plan will behave abnormally, too.[9] If $k = 0$, this rule establishes a health state as a single cause for action failure.

The intuitive idea behind a causal diagnosis is to be able to explain a given plan diagnosis Q by a (usually smaller) set of qualifications (causes) Q' together with some health state h of the agent established at time t using the set of causal rules R. Using such a pair consisting of a health state and a qualification should enable us to generate, using the rules in R, a set containing Q.

To define the effect of applying R to a set of (unique) instances of actions occurring in a plan, we first construct the set $inst(R)$ of instance of actions with respect to given plan $P = \langle \mathcal{A}, A, < \rangle$ as follows:

[9] We allow abnormal health states to be detected at the same time that abnormal action consequences are generated.

– For every rule r of the form $(\alpha_1, \alpha_2, \ldots, \alpha_k) \rightarrow \alpha_{k+1} \in R$, $inst(R)$ contains an instance $(a_{i_1}, a_{i_2}, \ldots, a_{i_k}) \rightarrow a_{i_{k+1}}$ of r whenever there exists a $t \geq 0$ such that $\{a_{i_1}, a_{i_2}, \ldots, a_{i_k}\} \subseteq P_{\leq t}$ and $a_{i_{k+1}} \in P_{>t}$.
– For every rule r of the form $(h; \alpha_1, \alpha_2, \ldots, \alpha_k) \rightarrow \alpha_{k+1} \in R$, $inst(R)$ contains the instances $(h; a_{i_1}, a_{i_2}, \ldots, a_{i_k}) \rightarrow a_{i_{k+1}}$, whenever there exists a $t \geq 0$ such that $\{a_{i_1}, a_{i_2}, \ldots, a_{i_k}\} \subseteq P_{\leq t}$ and $a_{i_{k+1}} \in P_{>t}$.

For each $r \in inst(R)$, let $ante(r)$ denote the antecedent of r and $hd(r)$ denote the head of r. Furthermore, let $Ab \subseteq \{h\}$ be a set containing an abnormal agent health state h or be equal to the empty set (signifying a normal state of the agent) and let $Q \subseteq A$ be a qualification of instances of actions. We can now define a causal consequence of a qualification Q and a health state Ab using R as follows:

Definition 3. *An instance $a \in A$ is a causal consequence of a qualification $Q \subset A$ and the health state Ab using the causal rules R if*

1. *$a \in Q$ or*
2. *there exists a rule $r \in inst(R)$ such that*
   (a) *for each $a_i \in ante(r)$ either a_i is a causal consequence of Q or $a_i \in Ab$, and*
   (b) *$a = hd(r)$.*

The set of causal consequences of Q using R and Ab is denoted by $C_{R,Ab}(Q)$.

We have a simple characterization of the set of causal consequences $C_{R,Ab}(Q)$ of a qualification Q and a health state Ab using a set of causal rules R:

Observation 1 $C_{R,Ab}(Q) = Cn_A(inst(R) \cup Q \cup Ab)$.

Here, $Cn_A(X)$ restricts the set of the set of classical consequences of a set of propositions X to the set $Lit(A)$. To avoid cumbersome notation, we will omit the subscripts R and Ab from the operator C and use $C(Q)$ to denote the set of consequences of a qualification Q using a health state Ab and a set of causal rules R.

We say that a qualification Q is closed under the set of rules R and an agent health state Ab if $Q = C(Q)$, i.e, Q is saturated under application of the rules R.

Proposition 1. *The operator C satisfies the following properties:*

1. *(inclusion): for every $Q \subseteq A$, $Q \subseteq C(Q)$*
2. *(idempotency): for every $Q \subseteq A$, $C(Q) = C(C(Q))$*
3. *(monotony): if $Q \subseteq Q' \subseteq A$ then $C(Q) \subseteq C(Q')$*

Proof. Note that $C(Q) = Cn(inst(R) \cup Q \cup Ab) \cap A$. Hence, monotony and inclusion follow immediately as a consequence of the monotony and inclusion of Cn. Monotony and inclusion imply $C(Q) \subseteq C(C(Q))$. To prove the reverse inclusion, let $Cn^*(Q) = Cn(instr(R) \cup Q \cup Ab)$. Then by inclusion and idempotency of Cn we have

$$C(C(Q)) = Cn^*(C(Q)) \cap A \ \subseteq \ Cn^*(Cn^*(Q)) \cap A = Cn^*(Q) \cap A = C(Q)$$

□

Thanks to Proposition 1 we conclude that every qualification can be easily extended to a closed set $C(Q)$ of qualifications. Due to the presence of causal rules, we require every diagnosis Q to be closed under the application of rules, that is, in the sequel we restrict diagnoses to closed sets $Q = C(Q)$.

Now we define a causal diagnosis as a qualification Q such that its set of consequences $C(Q)$ constitutes a diagnosis:

Definition 4. *Let $P = \langle \mathcal{A}, A, < \rangle$ be a plan, R a set of causal rules and let $obs(t)$ and $obs(t')$ be two observations with $t < t'$. Then a qualification $Q \subseteq A$ is a causal Ab-diagnosis of $(P, obs(t), obs(t'))$ if $C(Q) \cap P_{[t;t']}$ is a diagnosis of $(P, obs(t), obs(t'))$.*

Like we defined a minimum diagnosis, we now define two kinds of minimum causal diagnoses: a minimum causal *set* diagnosis and a minimum causal *effect* diagnosis:

Definition 5. *Let $P = \langle \mathcal{A}, A, < \rangle$ be a plan and $obs(t)$ and $obs(t')$ with $t < t'$ be two observations.*

1. *A* minimum causal set diagnosis *is a causal diagnosis Q such that $|Q| \leq |Q'|$ for every causal diagnosis Q' of P;*
2. *A* minimum causal effect diagnosis *is a causal diagnosis Q such that $|C(Q)| \leq |C(Q')|$ for every causal diagnosis Q'.*

Maximum informative causal set and maximum informative causal effect diagnoses are defined completely analogous to the previous definitions using standard diagnosis.

The relationships between the different diagnostic concepts we have distinguished is partially summarized in the following proposition:

Proposition 2. *Let $P = \langle \mathcal{A}, A, < \rangle$ be a plan and $obs(t)$ and $obs(t')$ with $t < t'$ be two observations.*

1. *$|Q| \leq |Q'|$ for every minimum causal set diagnosis Q and minimum closed diagnosis Q' of P;*
2. *$|Q| \leq |Q'|$ for every minimum causal effect diagnosis Q and minimum closed diagnosis Q' of P*

Proof. Both properties follow immediately from the definitions and the inclusion property of C. □

5.2 Causal diagnoses and Prediction

Except for playing a role in establishing causal *explanations* of observations, (causal) diagnoses also can play a significant role in the *prediction* of future results (states) of the plan or even the attainability of the goals of the plan. First of all, we should realize that a diagnosis can be used to enhance observed state information as follows: Suppose that Q is a causal Ab-diagnosis of a plan P based on the observations $obs(t)$ and $obs(t')$ for some $t < t'$, let $obs(t) \rightarrow^*_{C(Q);P} (\pi'_Q, t')$ and let $obs(t') = (\pi', t')$. Since $C(Q)$ is a diagnosis, π' and π'_Q agree upon the values of all objects occurring in both states. Therefore we can combine the information contained in both partial states by merging them into a new partial state $\pi'_{\sqcup} = \pi'_Q \sqcup \pi'$. Here, the merge $\pi^1 \sqcup \pi^2$ of two partial states π^1 and π^2 is simply defined as the partial state π where $\pi_j = \pi^i_j$ iff π^i_j is defined for $i = 1, 2$ and undefined else. $\pi'_{\sqcup}$ can be seen as the partial state that can be obtained by direct observation at time t and by making use of previous observations and plan information.

In the same way, we can use this information and the causal consequences $C(Q)$ to derive a prediction of the partial states derivable at times $t'' > t'$:

Definition 6. *Let Q is a causal Ab-diagnosis of a plan P based on the observations (π, t) and (π', t') where $t < t'$. Furthermore, let $obs(t) \rightarrow^*_{C(Q);P} (\pi'_Q, t')$ and let $obs(t') = (\pi', t')$. Then, for some time $t'' > t'$, (π'', t'') is the partial state predicted using Q and the observations if $(\pi'_Q \sqcup \pi', t') \rightarrow^*_{C(Q);P} (\pi'', t'')$.*

In particular, if $t'' = depth(P)$, i.e., the plan has been executed completely, we can predict the values of some objects that will result from executing P and we can check which goals $g \in G$ will still be achieved by the execution of the plan, based on our current knowledge. That is, we can check for which goals $g \in G$ it holds that $\tau \models g$. So causal diagnosis might also help in evaluating which goals will be affected by failing actions.

5.3 Complexity and implementation issues

It is well-known that the diagnosis problem is computationally intractable. The decision forms of both consistency-based and abductive based diagnosis are NP-hard ([2]). It is easy to see that standard plan diagnosis has the same order of complexity. Concerning (minimal) causal diagnoses, we can show that they are not more complex than establishing plan diagnoses if the latter problem is NP-hard. The reason is that in every case the verification of Q' being an Ab-causal diagnosis is as difficult as verifying a plan diagnosis under the assumption that the set $inst_P(R)$ is polynomially bounded in the size $||P||$ of the plan P.[10] Also note that subset minimality (under a set of rules $inst(R)$ of a set of causes can be checked in polynomial time.

The implementation of the diagnostic process is rather straight forward (see for instance [13]). First, we have to predict the expected result of the plan keeping of the actions involved in establishing the value of each object. Second, we determine which of the predicted values conflict with observed values resulting in conflict sets. Third, we have to solve a minimal hitting set problem given the conflict sets.

6 Conclusion

We have presented a new object-oriented model to specify plans and to apply techniques developed for model-based agent diagnosis. We distinguished two types of diagnosis: minimum plan diagnosis and maximum informative diagnosis to identify (*i*) minimum sets of anomalously executed actions and (*ii*) maximum informative (w.r.t. to predicting

[10] The reason is that computing consequences of Horn-theories can be achieved in a time linear in the size of $inst_P(R)$.

the observations) sets of anomalously executed actions. Assuming that a plan is carried out by a single agent, anomalously executed actions can be correlated if the anomaly is caused by some malfunctions in the agent. Therefore, (*iii*) causal diagnoses have been introduced and we have extended the diagnostic theory enabling the prediction of future failure of actions.

Current work can be extended in several ways. We mention two possible extensions:

First of all, we could improve the diagnostic model of the executing agent. The causal diagnoses are based on the assumption that the agent enters an abnormal state at some time point and stays in that state until the agent is repaired. In our future work we wish to extend the model such that the agent might evolve through several abnormal states. The resulting model will be related to diagnosis in Discrete Event Systems [7, 12]. Moreover, we intend to investigate plan repair in the context of the agent's current (abnormal) state.

Secondly, we would like to extend the diagnostic model with sequential observations and iterative diagnoses. Here, we would like to consider the possibilities of diagnosing a plan if more than two subsequent observations are made, the best way to detect errors in such cases and the construction of enhanced prediction methods.

References

1. L. Birnbaum, G. Collins, M. Freed, and B. Krulwich. Model-based diagnosis of planning failures. In *AAAI 90*, pages 318–323, 1990.
2. T. Bylander, D. Allemang, M.C. Tanner, and J.R. Josephson. The computational complexity of abduction. *Artificial Intelligence*, 49(1-3):25–60, 1991.
3. N. Carver and V.R. Lesser. Domain monotonicity and the performance of local solutions strategies for cdps-based distributed sensor interpretation and distributed diagnosis. *Autonomous Agents and Multi-Agent Systems*, 6(1):35–76, 2003.
4. L. Console and P. Torasso. Hypothetical reasoning in causal models. *International Journal of Intelligence Systems*, 5:83–124, 1990.
5. L. Console and P. Torasso. A spectrum of logical definitions of model-based diagnosis. *Computational Intelligence*, 7:133–141, 1991.
6. F. de Jonge and N. Roos. Plan-execution health repair in a multi-agent system. In *PlanSIG 2004*, 2004.
7. R. Debouk, S. Lafortune, and D. Teneketzis. Coordinated decentralized protocols for failure diagnosis of discrete-event systems. *Journal of Discrete Event Dynamical Systems: Theory and Application*, 10:33–86, 2000.
8. R. E. Fikes and N. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 5:189–208, 1971.
9. Bryan Horling, Brett Benyo, and Victor Lesser. Using Self-Diagnosis to Adapt Organizational Structures. In *Proceedings of the 5th International Conference on Autonomous Agents*, pages 529–536. ACM Press, 2001.
10. M. Kalech and G. A. Kaminka. On the design ov social diagnosis algorithms for multi-agent teams. In *IJCAI-03*, pages 370–375, 2003.
11. M. Kalech and G. A. Kaminka. Diagnosing a team of agents: Scaling-up. In *AAMAS 2004*, 2004.
12. Y. Pencolé, M. Cordier, and L. Rozé. Incremental decentralized diagnosis approach for the supervision of a telecommunication network. In *DX01*, 2001.
13. R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

Intention recognition in the Situation Calculus and Probability Theory frameworks

Robert Demolombe[1] and Ana Mara Otermin Fernandez[2] *

[1] ONERA Toulouse
France
Robert.Demolombe@cert.fr
[2] ONERA Toulouse
France
txantxita@hotmail.com

Abstract. A method to recognise agent's intentions is presented in a framework that combines the logic of Situation Calculus and Probability Theory. The method is restricted to contexts where the agent only performs procedures in a given library of procedures, and where the system that intends to recognise the agent's intentions has a complete knowledge of the actions performed by the agent.
An original aspect is that the procedures are defined for human agents and not for artificial agents. The consequence is that the procedures may offer the possibility to do any kind of actions between two given actions, and they also may forbid to perform some specific actions. Then, the problem is different and more complex than the standard problem of plan recognition.
To select the procedures that partially match the observations we consider the procedures that have the greatest estimated probability. This estimation is based on the application of Bayes' theorem and on specific heuristics. These heuristics depend on the history and not just on the last observation.
A PROLOG prototype of the presented method has been implemented.

1 Introduction

When two agents have to interact it is important for each agent to know the other agent's intentions because this knowledge allows to anticipate his future behavior. This information can be used either to help the other agent to do what he intends to do or to control whether what he does is compatible with his intention. Even if an agent can never be sure that he knows the other agent's intentions an uncertain information is much better than a complete ignorance when a decision has to be taken.

In this paper a method is proposed to recognise what are the agent's intentions in the particular context of a pilot that interacts with an aircraft. The first specificity of this context is that the pilot performs procedures that are very well

* Also student at: Universidad Politenica de Madrid.

defined in a handbook. The second specificity is that the procedures are defined in terms of commands that have to be performed (like to turn a switch on) and it is reasonable to assume that the performance of these commands can be perceived thanks to sensors in the aircraft. Then, it is possible to design a system that has the capacity to observe all the commands performed by the pilot.

Under this assumption the system can compare the sequence of observations with the procedure definitions in the handbook and it can determine the procedures that match with these observations. The procedures that have the "best" match are assigned to the agent's intentions.

To define a method to recognise the pilot's intentions we have to find solutions to three independent problems:

1. to select a language to represent the procedures in formal terms,
2. to define a formal characterisation of the procedures that match with the observations,
3. to define a method to select the procedures that have the "best" match and are assigned to the agent's intention.

In a previous work Demolombe and Hamon [9] have proposed solutions to problems 1 and 2 in the logical framework of the Situation Calculus. The Situation Calculus is a variant of classical first order logic, that is the reason why it is more convenient for computational logic than modal logics.

The contribution of this paper is to propose a solution to problem 3 in a framework that combines Situation Calculus and Probability Theory and which is based on Bayes' theorem.

There are many other works that have similar objectives in the field of plan recognition [11] and many of them make use of probabilities [4, 7, 1, 5] or use an utility function [13]. Baier in [3] also uses the framework of the Situation Calculus but without probabilities. Many of them have been designed in the particular context of natural language analysis [6, 2, 5] or game theory [1].

The original feature in our case is that the pilot's procedures may allow any other command in between a sequence of two prescribed commands and it may be specified that some commands are forbidden. Also it may happen that the pilot has the intention to perform several procedures in parallel. The consequence is that problems 2 and 3 are much more complex than the standard problem of plan recognition.

The paper is organised as follows. In sections 2 and 3 the solutions to problems 1 and 2 are recalled. In section 4 the method to solve problem 3 is presented. In that section we start with the analysis of a typical example, we define a general method to compute probabilities, we define heuristics to estimate the probabilities and finally we apply the method to the example to show that the results fit the intuitive requirements. Possible refinements or extensions of the method are presented in the conclusion.

Since the method can be applied to many other contexts we shall use the general term "agent" instead of "pilot", and "action" instead of "command".

2 A brief introduction to the Situation Calculus and to a GOLOG extension

The logical framework of Situation Calculus [15] is used to represent the states of the world and the actions that are performed by the agent.

The Situation Calculus is a typed first order logic with equality (except some limited fragments that are of second order). In the language there are two kinds of predicates. The predicates whose truth value may change after performance of an action are called "fluents". They have exactly 1 argument of the type situation which is the last argument. The other predicates have no argument of the type situation.

For example, we may have the predicates:
$nationality(x)$: the nationality of the aircraft is x.
$gear.extended(s)$: in the situation s the landing gear is extended.
$altitude(x, s)$: in the situation s the aircraft altitude is x.

Here $altitude(x, s)$ and $gear.extended(s)$ are fluents, and $nationality(x)$ is not a fluent.

The terms of type situation may be constant or variable symbols of the type situation, or terms of the form $do(a, s)$ where do is a designated function symbol, a is a term of type action and s is a term of type situation.

For instance, if S_0 is a constant of type situation and $extend.gear$ and $retract.gear$ are constants of type action, the following terms are of type situation: S_0, $do(extend.gear, S_0)$, $do(extend.gear, s)$ and $do(retract.gear, do(extend.gear, S_0))$.

The term $do(retract.gear, do(extend.gear, S_0))$ denotes the situation where we are after performance of the actions $extend.gear$ and $retract.gear$.

As a matter of simplification we use the notation $do([a_1, \dots, a_n], s)$ to denote $do(a_n, \dots, do(a_1, s) \dots)$.

The grammar of the formulas of the Situation Calculus is defined as usual for classical first order logics.

A successor relation [3] is defined on the set of situations. Intuitively $s \leq s'$ means that the situation s' is reached from the situation s after some sequence of action. In semiformal terms, $s \leq s'$ is the smallest relation that satisfies the following properties:

$s \leq s' \overset{\text{def}}{=} (s < s') \vee (s = s')$
$\forall s \forall s' \forall a (s' = do(a, s) \rightarrow s < s')$
$\forall s \forall s' \forall s'' ((s < s') \wedge (s' < s'') \rightarrow (s < s''))$

To define the truth value of the fluents in any situation a successor state axiom has to be given for each fluent. For example, for $gear.extended(s)$ we have:

$\forall s \forall a (gear.extended(do(a, s)) \leftrightarrow a = extend.gear \vee gear.extended(s) \wedge \neg(a = retract.gear))$

[3] In this paper the definition of the successor relation is the only part of the Situation Calculus that requires second order logic.

The intuitive meaning of this axiom is that the only action that can cause $gear.extended(do(a, s))$ to be true (resp. false) is the action $extend.gear$ (resp. $retract.gear$).

The GOLOG language [12] is a programming language for robots but it can be used for other kinds of agents. Its expressive power is the same as ALGOL and its semantics is defined in the logic of the Situation Calculus. Programs are terms that represent complex actions defined with several operators.

Here, for simplicity, we have only considered the operator of sequence (denoted by ";"), test (denoted by "ϕ?") and non deterministic choice (denoted by "|"). To represent what is called in the following "procedures" we have added the "negation" operator (denoted by "$-$") and the "any sequence of actions" term (denoted by "σ"). The motivation of this extension can be explained with the following example.

Let us, consider the procedure called "fire on board", which is described for a small private aircraft. The procedure says that in case of engine fire the pilot 1) turns off fuel feed, 2) sets full throttle, and 3) sets mixture off. These three primitive actions, or commands, are respectively denoted by $fuel.off$, $full.throttle$ and $mixture.off$, and the procedure is denoted by $fire.on.board$.

However, it is implicit in the procedure definition that between actions 1) and 2) or between 2) and 3) the pilot can do any other action. For example, he can call air traffic control. It is also implicit that after turning off fuel feed he must not turn on fuel feed. That is just common sense for a human being but it has to be made explicit to define a formal method that can be used by the system which observes the pilot.

Then, in the modified GOLOG language the "fire on board" procedure is represented by:
$fire.on.board \overset{\text{def}}{=} fuel.off; (\sigma/fuel.on); full.throttle; (\sigma/fuel.on); mixture.off$
where α_1/α_2 is an abbreviation for $\alpha_1 - (\sigma; \alpha_2; \sigma)$ which intuitively means that the sequence of actions which is a performance of α_1 must not contain a sequence of actions which is a performance of α_2.

In the case of programs for an artificial agent there is no need for the term σ nor for the operator "/" because **an artificial agent only does what is specified in the program**. That makes the basic difference between a program and what is called here a "procedure".

The formal definition of the modified GOLOG language is :

- atomic actions, test actions and σ are procedures,
- if α_1 and α_2 are procedures, then $(\alpha_1; \alpha_2)$, $(\alpha_1|\alpha_2)$ and $(\alpha_1 - \alpha_2)$ are procedures.

The formal definition of the procedures is defined by formulas of the Situation Calculus language. These formulas are denoted by the property $Do_p(\alpha, s, s')$ whose intuitive meaning is:

$Do_p(\alpha, s, s')$: s' is a situation that can be reached from the situation s after performance of the procedure α.

The formal semantics of $Do_p(\alpha, s, s')$ is:

$Do_p(a, s, s') \overset{\text{def}}{=} s' = do(a, s)$ if a is an atomic action.

$Do_p(\sigma, s, s') \overset{\text{def}}{=} s \leq s'$

$Do_p(\phi?, s, s') \overset{\text{def}}{=} \phi[s] \wedge s' = s$

$Do_p(\alpha_1; \alpha_2, s, s') \overset{\text{def}}{=} \exists s_1(Do_p(\alpha_1, s, s_1) \wedge Do_p(\alpha_2, s_1, s'))$

$Do_p(\alpha_1 | \alpha_2, s, s') \overset{\text{def}}{=} Do_p(\delta_1, s, s') \vee Do_p(\delta_2, s, s')$

$Do_p(\alpha_1 - \alpha_2, s, s') \overset{\text{def}}{=} Do_p(\alpha_1, s, s') \wedge \neg Do_p(\alpha_2, s, s')$

This modified GOLOG language gives a solution to the problem 1 that we have mentioned in the introduction.

3 Doing a procedure

To characterise the fact that a sequence of performed actions "matches" a partial performance of a procedure, in the sense that this sequence **can be interpreted** as a partial performance of the procedure, we use the property $Doing(\alpha, s, s')$. However, this property does not guarantee that the agent is performing this procedure.

In informal terms the property $Doing(\alpha, s, s')$ holds if the three following conditions are satisfied:

1. The agent has begun executing a part α' of α between s and s'.
2. The agent has not completely executed α between s and s'.
3. The actions performed between s and s' do not prevent the continuation of the execution of α.

In a first step we define the property $Do_m(\alpha, s, s')$ whose intuitive meaning is that we have $Do_p(\alpha, s, s')$ and there is no shorter sequence of actions between s and s' such that we have Do_p for this sequence. We have:

$Do_m(\alpha, s, s') \overset{\text{def}}{=} Do_p(\alpha, s, s') \wedge \neg\exists s_1(Do_p(\alpha, s, s_1) \wedge s_1 \leq s')$

Then, we define the property $Do_s(\alpha, s, s')$ whose intuitive meaning is that the sequence of actions between s and s' satisfies the above conditions 1, 2 and 3. We have:

$Do_s(\alpha, s, s') \overset{\text{def}}{=} \exists \alpha'(start(\alpha', \alpha) \wedge$
$\exists s_1(s_1 \leq s' \wedge Do_m(\alpha', s, s_1)) \wedge$
$\neg\exists s_2(s_2 \leq s' \wedge Do_m(\alpha, s, s_2)) \wedge$
$\exists s_3(s' < s_3 \wedge Do_m(\alpha, s, s_3)))$

where $start(\alpha', \alpha)$ means that α can be reformulated into a procedure of the form: $(\alpha'; \alpha'')|\beta$ which has the same semantics as α, i.e. $\forall s \forall s'(Do_p(\alpha, s, s') \leftrightarrow Do_p((\alpha'; \alpha'')|\beta, s, s'))$.

The condition 1 is expressed by $\exists \alpha'(start(\alpha', \alpha) \wedge \exists s_1(s_1 \leq s' \wedge Do_m(\alpha', s, s_1))$, the strict interpretation of condition 2 is expressed by $\neg\exists s_2(s_2 \leq s' \wedge Do_m(\alpha, s, s_2))$, and the condition 3 is expressed by $\exists s_3(s' < s_3 \wedge Do_m(\alpha, s, s_3))$.

Finally, the definition of $Doing(\alpha, s, s')$ is:

$Doing(\alpha, s, s') \overset{\text{def}}{=} \exists s_1(s \leq s_1 \wedge Do_s(\alpha, s_1, s')) \wedge \neg\exists s_2(s \leq s_2 \wedge s_2 < s_1 \wedge Do_s(\alpha, s_2, s_1)))$

The condition $\exists s_1(s \leq s_1 \wedge Do_s(\alpha, s_1, s'))$ expresses that there is an execution of α that has begun in s_1 and has not ended, and the condition $\neg\exists s_2(s \leq s_2 \wedge s_2 < s_1 \wedge Do_s(\alpha, s_2, s_1))$ expresses that there is no previous α execution which has started and not ended before s_1.

4 Intention recognition

This section presents a method for choosing between several procedures, that satisfy the *Doing* property, the one that can be assigned by the system to the agent's intention.

This assignment is never guaranteed to correspond to the true agent's intention, and due to this uncertainty it is sensible to make use of probabilities to make the choice.

Before going into the formal presentation of the method let us give a simple example to intuitively show what are the basic principles [4] and assumptions of the method.

4.1 A simple example

Let us consider the three following procedures.
$\alpha = a; \sigma; b; \sigma; c$
$\beta = d; \sigma; e$
$\gamma = a; \sigma; f$

Let us assume that we are in the situation s_5 where the following sequence of actions has been performed: $[f, a, d, b, c]$, that is in formal terms:
$s_5 = do([f, a, d, b, c], s_0)$.

In the situation $s_1 = do(f, s_0)$ there is no procedure which is compatible with the performed action f. We have $\neg Doing(\alpha, s_0, s_1)$, $\neg Doing(\beta, s_0, s_1)$ and $\neg Doing(\gamma, s_0, s_1)$.

We have adopted the following assumption.

Assumption H1. If an agent has the intention to do a procedure α then he does the actions that are defined by the procedure α.

According to H1 in s_1 the system knows that the agent did not have the intention to do α in s_0, because if he had the intention to do α in s_0 he would have started to do α and he would have done the action a in s_1 instead of f. The same for β and γ.

Nevertheless in s_0 the system can accept that the probability that the agent has the intention to do α is not equal to 0. Then, we have accepted the additional assumption:

Assumption H2. If the agent in the situation s_i is not doing α, in the sense that $\neg Doing(\alpha, s_0, s_i)$, then in s_i the probability that he has the intention to do α is independent of s_i, and this probability is denoted by $\pi(\alpha)$.

[4] These principles are expected properties and they should not be confused with the assumptions.

Let us define the following notations.

$P(\phi)$: probability that ϕ holds.

$Int(\alpha, s_i)$: in the situation s_i the agent has the intention to do α [5].

In formal terms H2 can be expressed by:

$\forall s \forall s' \forall \alpha (s \leq s' \wedge \neg Doing(\alpha, s, s') \rightarrow P(Int(\alpha, s')) = \pi(\alpha))$

Since for any procedure α we have $\neg Doing(\alpha, s_0, s_0)$, from H2 we have: $P(Int(\alpha, s_0)) = P(Int(\alpha, s_1)) = \pi(\alpha)$, $P(Int(\beta, s_0)) = P(Int(\beta, s_1)) = \pi(\beta)$ and $P(Int(\gamma, s_0)) = P(Int(\gamma, s_1)) = \pi(\gamma)$.

In the situation $s_2 = do([f, a], s_0)$ we have $\neg Doing(\beta, s_0, s_2)$ and $P(Int(\beta, s_2)) = \pi(\beta)$, and now we have $Doing(\alpha, s_0, s_2)$ and $Doing(\gamma, s_0, s_2)$.

The fact that the action a has been performed is a good argument for the system to believe that the agent has the intention to do α and to believe that he has the intention to do γ. Then we should have $P(Int(\alpha, s_2)) > P(Int(\alpha, s_1))$ and $P(Int(\gamma, s_2)) > P(Int(\gamma, s_1))$.

It is sensible to assume that $P(Int(\alpha, s_i))$ and $P(Int(\gamma, s_i))$ increase in the same way from s_1 to s_2.

So, if $\pi(\alpha) = \pi(\beta) = \pi(\gamma)$, $Int(\alpha, s_2)$ and $Int(\gamma, s_2)$ have the same and the greatest probability and the system believes that the agent has the intention to do α and that he has the intention to do γ.

Let us use the following notation.

$BInt(\alpha, s_i)$: in the situation s_i the system believes that the agent has the intention to do α.

Using this notation we have: $BInt(\alpha, s_2)$, $\neg BInt(\beta, s_2)$ and $BInt(\gamma, s_2)$.

We have adopted the following general assumption.

Assumption H3. In a situation s_i such that $Doing(\alpha, s_0, s_i)$, if there is no procedure β such that $Doing(\beta, s_0, s_i)$ and $P(Int(\beta, s_i)) > P(Int(\alpha, s_i))$, then the system believes in s_i that the agent has the intention to do α (i.e. we have $BInt(\alpha, s_i)$).

H3 can be reformulated as:

$BInt(\alpha, s)$ iff $Doing(\alpha, s_0, s)$ and there is no procedure β such that $P(Int(\beta, s)) > P(Int(\alpha, s))$

In the situation $s_3 = do([f, a, d], s_0)$ we have $Doing(\alpha, s_0, s_3)$, $Doing(\beta, s_0, s_3)$ and $Doing(\gamma, s_0, s_3)$.

In s_3 we can assume that $P(Int(\beta, s_i))$ has increased from s_2 to s_3 in the same way as $P(Int(\alpha, s_i))$ and $P(Int(\gamma, s_i))$ have increased from s_1 to s_2.

For the procedures α and γ, in s_2 the agent has the choice between doing the next recommended action (that are respectively b and f) or doing any other action. We have assumed that if he does not do the recommended action, then the probability to do the corresponding procedure decreases, because the last observed action does not confirm that he has the intention to do this procedure.

Then, if $\pi(\alpha) = \pi(\beta) = \pi(\gamma)$ we have: $P(Int(\alpha, s_3)) < P(Int(\beta, s_3))$ and $P(Int(\gamma, s_3)) < P(Int(\beta, s_3))$, and therefore we have $BInt(\beta, s_3)$, $\neg BInt(\alpha, s_3)$ and $\neg BInt(\gamma, s_3)$).

[5] To be more precise we should say that the agent has the intention to reach a situation where α has been done.

In the situation $s_4 = do([f, a, d, b], s_0)$ we have $Doing(\alpha, s_0, s_4)$, $Doing(\beta, s_0, s_4)$ and $Doing(\gamma, s_0, s_4)$.

In that situation the action b is a recommended action for α but it is not a recommended action for γ. Then, if $\pi(\alpha) = \pi(\gamma)$ we should have $P(Int(\alpha, s_4)) > P(Int(\gamma, s_4))$.

If we compare the procedures α and β in s_4, there are two performed actions (a and b) that are recommended in α, and there is only one (a) which is recommended in β. The number of performed actions that are not recommended is the same for α and β (action d for α and action b for β). Therefore, if $\pi(\alpha) = \pi(\beta)$ we should have $P(Int(\alpha, s_4)) > P(Int(\beta, s_4))$. Then, we have $BInt(\alpha, s_4)$, $\neg BInt(\beta, s_4)$ and $\neg BInt(\gamma, s_4)$.

In the situation $s_5 = do([f, a, d, b, c], s_0)$ we have $\neg Doing(\alpha, s_0, s_5)$, $Doing(\beta, s_0, s_5)$ and $Doing(\gamma, s_0, s_5)$.

The number of recommended actions is 1 for β and γ in s_5, but the number of not recommended actions is 3 for γ and 2 for β. Then, if $\pi(\alpha) = \pi(\beta) = \pi(\gamma)$ we should have $P(Int(\beta, s_5)) > P(Int(\gamma, s_5))$ and $P(Int(\beta, s_5)) > P(Int(\alpha, s_5))$. Therefore we have $BInt(\beta, s_5)$, $\neg BInt(\alpha, s_5)$ and $\neg BInt(\gamma, s_5)$.

From this example we can derive some general principles that are expressed with the following terminology.

In a procedure definition we call an action a **prescribed** action if that action explicitly appears in the procedure and it is just preceded by an explicit action.

For example, if α has the form: $\ldots; a; b; \ldots$ then this occurrence of b is a prescribed action in α. Notice that in a given procedure some occurrences of b may be prescribed actions and others not, like in $\alpha = c; \sigma; b; a; b$.

In a procedure definition we call an action a **recommended** action if that action explicitly appears in the procedure and it is just preceded by a term of the form σ or σ/β.

For example, if α has the form: $\ldots; \sigma; a; \ldots$ or $\ldots; \sigma/(b|c); a; \ldots$ then this occurrence of a is a recommended action in α.

Let us call A the set of actions that can be done by the agent and can be observed by the system.

In a procedure definition we call an action a **tolerated** action if the procedure has the form: $\ldots; \sigma; a; \ldots$ and this action is in $A - \{a\}$.

For example, if $A = \{a, b, c, d, e\}$ and α has the form: $\ldots; \sigma; a; \ldots$, then the set of tolerated actions for this occurrence of σ is $\{b, c, d, e\}$.

In a procedure definition we call an action a **restricted tolerated** action if the procedure has the form: $\ldots; \sigma/(a_{i_1}| \ldots |a_{i_l}); a; \ldots$ and this action is in $A - \{a_{i_1}, \ldots, a_{i_l}, a\}$.

For example, if α has the form: $\ldots; \sigma/(b|d); a; \ldots$ the set of restricted tolerated actions for this occurrence of σ is $\{c, e\}$.

With these definitions we can formulate our basic principles in that way.

Principle A. If in the situation s_i the last performed action is a prescribed action of α, then $P(Int(\alpha, s_i))$ should "strongly" increase with respect to $P(Int(\alpha, s_{i-1}))$.

Principle B. If in the situation s_i the last performed action is a recommended action of α, then $P(Int(\alpha, s_i))$ should increase with respect to $P(Int(\alpha, s_{i-1}))$, but it should increase less than in the case of a prescribed action.

Principle C. If in the situation s_i the last performed action is a tolerated action of α, then $P(Int(\alpha, s_i))$ should decrease with respect to $P(Int(\alpha, s_{i-1}))$.

Principle D. If in the situation s_i the last performed action is a restricted tolerated action of α, then the fact that $P(Int(\alpha, s_i))$ increases or decreases with respect to $P(Int(\alpha, s_{i-1}))$ depends on the cardinality of the set of restricted tolerated actions.

We also have adopted the following assumption about the evolution of the fact that the agent has the intention to do a procedure α.

Assumption H4. In a situation s_i such that we have $Doing(\alpha, s_0, s_i)$ it is assumed that the agent has in s_i the intention to do α iff he has the intention to do α in s_{i-1}.

The assumption H4 is expressed in formal terms as follows.

$(H4)$ $\forall s \forall s' \forall s'' \forall a \forall \alpha ((Doing(\alpha, s, s'') \wedge s'' = do(a, s')) \rightarrow (Int(\alpha, s'') \leftrightarrow Int(\alpha, s')))$

H4 is logically equivalent to the conjunction of H'4 and H"4.

$(H'4)$ $\forall s \forall s' \forall s'' \forall a \forall \alpha (Doing(\alpha, s, s'') \wedge s'' = do(a, s') \wedge Int(\alpha, s') \rightarrow Int(\alpha, s''))$

$(H''4)$ $\forall s \forall s' \forall s'' \forall a \forall \alpha (Doing(\alpha, s, s'') \wedge s'' = do(a, s') \wedge Int(\alpha, s'') \rightarrow Int(\alpha, s'))$

The assumption H'4 means that the agent's intention is persistent as long as the procedure α is not completely performed. That corresponds to the notion of intention persistence proposed by Cohen and Levesque in [8] (see also [14]).

The assumption H"4 corresponds to a different idea. This idea is that if the action a performed by the agent is consistent with the fact that he is doing α and in the situation s'' the agent has the intention to do α, then he has performed the action a **because** in s' he had the intention to do α.

4.2 General method to compute the probabilities

To present the general method we shall use the following notations.

$A = \{a_1, a_2, \ldots, a_N\}$: set of actions that can be performed by the agent and that can be observed by the system.

We adopt the following assumption.

Assumption H5. It is assumed that in the language definition the set of atomic action constant symbols is A.

The assumption H5 intuitively means that the actions performed by the agent that cannot be observed by the system are ignored by the system. This assumption is consistent with the fact that what the system believes about the agents' intentions is only founded on his observations.

o_i: ith observation action performed by the system.

$a_{j_i} = obs(o_i)$: a_{j_i} is the action performed by the agent that has been observed by the system by means of the observation action o_i.

O_i: short hand to denote the proposition $a_{j_i} = obs(o_i)$.

$O_{1,i} \overset{\text{def}}{=} O_1 \wedge O_2 \wedge \ldots \wedge O_i$

$O_{1,0} \overset{\text{def}}{=} true$

s_0: initial situation.

$s_i = do(a_{j_i}, s_{i-1})$

$P(Int(\alpha, s_i)|O_{1,i})$: probability that in the situation s_i the agent has the intention to do α if the sequence of observations is $O_{1,i}$.

From Bayes' theorem we have:

(1) $P(Int(\alpha, s_i)|O_{1,i}) = \frac{P(O_{1,i}|Int(\alpha,s_i))\times P(Int(\alpha,s_i))}{P(O_{1,i})}$

From (1) we have:

(2) $P(Int(\alpha, s_i)|O_{1,i}) = \frac{P(O_i \wedge O_{1,i-1}|Int(\alpha,s_i))\times P(Int(\alpha,s_i))}{P(O_i \wedge O_{1,i-1})}$

Then, we have:

(3) $P(Int(\alpha, s_i)|O_{1,i}) = \frac{P(O_i|O_{1,i-1} \wedge Int(\alpha,s_i))}{P(O_i|O_{1,i-1})} \times \frac{P(O_{1,i-1}|Int(\alpha,s_i))\times P(Int(\alpha,s_i))}{P(O_{1,i-1})}$

If $\neg Doing(\alpha, s_0, s_i)$**:**

From H2 we have: $P(Int(\alpha, s_i)|O_{1,i}) = P(Int(\alpha, s_i))$. Then we have:

(4) $P(Int(\alpha, s_i)|O_{1,i}) = \pi(\alpha)$

If $Doing(\alpha, s_0, s_i)$**:**

From H4 we have: $Int(\alpha, s_i) \leftrightarrow Int(\alpha, s_{i-1})$.

Then, from (3) we have:

(5) $P(Int(\alpha, s_i)|O_{1,i}) = \frac{P(O_i|O_{1,i-1} \wedge Int(\alpha,s_{i-1}))}{P(O_i|O_{1,i-1})} \times \frac{P(O_{1,i-1}|Int(\alpha,s_{i-1}))\times P(Int(\alpha,s_{i-1}))}{P(O_{1,i-1})}$

Therefore we have:

(6) $P(Int(\alpha, s_i)|O_{1,i}) = \frac{P(O_i|O_{1,i-1} \wedge Int(\alpha,s_{i-1}))}{P(O_i|O_{1,i-1})} \times P(Int(\alpha, s_{i-1})|O_{1,i-1})$

If we adopt the notations:

$num_i(\alpha) \overset{\text{def}}{=} P(O_i|O_{1,i-1} \wedge Int(\alpha, s_{i-1}))$

$den_i(\alpha) \overset{\text{def}}{=} P(O_i|O_{1,i-1})$

$F_i(\alpha) \overset{\text{def}}{=} \frac{num_i(\alpha)}{den_i(\alpha)}$

We have:

(7) $P(Int(\alpha, s_i)|O_{1,i}) = F_i(\alpha) \times P(Int(\alpha, s_{i-1})|O_{1,i-1})$

The formula (7) allows to regress the computation of $P(Int(\alpha, s_i)|O_{1,i})$ until a situation s_j where we have $\neg Doing(\alpha, s_0, s_j)$ [6].

4.3 Heuristics to estimate the probabilities

To define heuristics to estimate the value of $F_i(\alpha)$ we have restricted the set of procedures to procedures of the form:

$\alpha = A_1; \Sigma_1; \ldots; A_k; \Sigma_k; A_{k+1}; \ldots; A_s$

where each A_k denotes an atomic action in A and Σ_k either is absent or denotes a term of the form $\sigma/(a_{i_1}|\ldots|a_{i_l})$ where each a_{i_j} is in A and l may be equal to 0. This form will be called in the following: "linear normal form".

Notice that this form is not a too strong restricted form because a procedure can be transformed by repeatedly applying the transformation rule that transforms $\alpha_1; (\alpha_2|\alpha_3); \alpha_4$ into $(\alpha_1; \alpha_2; \alpha_4)|(\alpha_1; \alpha_3; \alpha_4)$. At the end we get a procedure in the form $\alpha = \alpha_1|\alpha_2|\ldots|\alpha_p$. Then, the only difference between each α_i and a procedure in linear normal form is that the A_ks may denote either an atomic

[6] Notice that for any procedure α we have $\neg Doing(\alpha, s_0, s_0)$.

action or a test action, and the Σ_ks, when they are not absent, have in general the form σ/β where β may be any kind of procedure.

Now we are going to define the estimation of the term $F_i(\alpha)$ in the case where we have $Doing(\alpha, s_0, s_i)$.

The estimation of $F_i(\alpha)$ depends on the part α'_{i-1} of α which has already been performed in the situation s_{i-1}. This part is defined by the property $Done(\alpha'_{i-1}, \alpha, s_0, s_i)$ where the property $Done$ is defined as follows.

$Done(\alpha', \alpha, s, s') \overset{\text{def}}{=} Doing(\alpha, s, s') \wedge start(\alpha', \alpha) \wedge \exists s_1(s \leq s_1 < s' \wedge Do_s(\alpha, s_1, s') \wedge Do_p(\alpha', s_1, s'))$

In this definition the condition $Do_s(\alpha, s_1, s')$ guarantees that the part of α that is being performed in s has started his performance in s_1, and the condition $Do_p(\alpha', s_1, s')$ guarantees that there is no part of α that is longer than α' that has been performed between s_1 and s'. $Done(\alpha', \alpha, s, s')$ intuitively means that α' is the maximal part of α that has started between s and s' and that has ended in s'.

For instance, in the previous example in s_2 we have $Doing(\alpha, s_0, s_2)$ and for $\alpha'_2 = a$ we have $Do_s(\alpha'_2, s_1, s_2)$ and $Do_p(\alpha'_2, s_1, s_2)$. In s_3 we have $\alpha'_3 = a; \sigma$ and in s_4 we have $\alpha'_4 = a; \sigma; b$.

To estimate $F_i(\alpha)$ we have accepted the following assumption.

Assumption H6. It is assumed that the ith observation 0_i is independent of the previous observations and each action in A has the same probability to be observed.

In formal terms H6 is expressed by: $den_i(\alpha) = P(O_i|O_{1,i-1}) = P(O_i) = \frac{1}{N}$.

We shall use the notation $O_i = A_k$ to express that the action a_{i_j} observed by the observation action o_i is the atomic action denoted by A_k, and we use the notation $O_i \in \Sigma_k$ to express that a_{i_j} is in the set $A - \{a_{i_1}, \ldots, a_{i_l}, a_{k+1}\}$, where a_{k+1} is the action denoted by A_{k+1}.

The terms $num_i(\alpha)$ and $F_i(\alpha)$ have to be estimated only in the case where we have $Doing(\alpha, s_0, s_i)$. We have to consider different cases.

Case 1. We have $\neg Doing(\alpha, s_0, s_{i-1})$.

In that case $\alpha'_{i-1} = A_1$ and, from the assumption H1, $Int(\alpha, s_{i-1})$ and $Doing(\alpha, s_0, s_i) \wedge \neg Doing(\alpha, s_0, s_{i-1})$ imply that in s_i the agent has performed the action A_1, and the observed action in O_i is A_1. Then, we necessarily have $O_i = A_1$.

Therefore we have $num_i(\alpha) = 1$ and $F_i(\alpha) = N$.

Case 2. We have $Doing(\alpha, s_0, s_{i-1})$.

- **Case 2.1.** α'_{i-1} has the form $\alpha'_{i-1} = \ldots; A_k$.
    - **Case 2.1.1.** α has the form
      $\alpha = \ldots; A_k; A_{k+1}; \ldots$.
      In that case Σ_k is absent in α. From the assumption H1, $Int(\alpha, s_{i-1})$ implies that the action performed in s_i is A_{k+1}. Then, we necessarily have $O_i = A_{k+1}$.
      Therefore we have $num_i(\alpha) = 1$ and $F_i(\alpha) = N$.

- **Case 2.1.2.** α has the form
  $\alpha = \ldots; A_k; \Sigma_k; A_{k+1}; \ldots$.
  Case 2.1.2.1. $O_i = A_{k+1}$.
  The general form of Σ_k is $\sigma/(a_{i_1}|\ldots|a_{i_l})$.
  According to principle B it is much more likely that the action performed by the agent in s_i is the recommended action A_{k+1} than any restricted tolerated action defined by Σ_k.
  Then we have $num_i(\alpha) = 1 - \epsilon$ where the value of ϵ is defined in function of the application domain and is supposed to be "small" with respect to 1.
  We have $F_i(\alpha) = N \times (1 - \epsilon)$.
  Case 2.1.2.2. $O_i \neq A_{k+1}$.
  Here we have adopted the following assumption.
  Assumption H7. It is assumed that when the agent has the intention to do α all the restricted tolerated actions have the same probability to be performed by the agent.
  According to H7 any action in $A - \{a_{i_1}, \ldots, a_{i_l}, a_{k+1}\}$ has the same probability to be done. Then, we have[7]: $num_i(\alpha) = \frac{\epsilon}{N-(l+1)}$.
  We have $F_i(\alpha) = \frac{N}{N-(l+1)} \times \epsilon$.

- **Case 2.2.** α'_{i-1} has the form $\alpha'_{i-1} = \ldots; \Sigma_k$.
  Case 2.2.1. $O_i = A_{k+1}$.
  We are in the same type of situation as in the case 2.1.2.1. Then we have $num_i(\alpha) = 1 - \epsilon$ and $F_i(\alpha) = N \times (1 - \epsilon)$.
  Case 2.2.2. $O_i \neq A_{k+1}$.
  We are in the same type of situation as in the case 2.1.2.2. Then we have $num_i(\alpha) = \frac{\epsilon}{N-(l+1)}$ and $F_i(\alpha) = \frac{N}{N-(l+1)} \times \epsilon$.

In the case where the action that has been performed by the agent in s_i is a prescribed action (cases 1. and 2.1.1.) we have $F_i(\alpha) = N$. This conforms the principle A.

In the case where the performed action is a recommended action (cases 2.1.2.1. and 2.2.1.) we have $F_i(\alpha) = N \times (1 - \epsilon)$. To fulfill the principle B, that is: $F_i(\alpha) > 1$, we have to assign to ϵ a value such that $\epsilon < \frac{N-1}{N}$.

In the case where the performed action is a tolerated action (cases 2.1.2.2. and 2.2.2. and $l = 0$) we have $F_i(\alpha) = \frac{N}{N-1} \times \epsilon$. From the assumption $\epsilon < \frac{N-1}{N}$ we have $F_i(\alpha) < 1$ and this fulfills the principle C.

In the case where the performed action is a restricted tolerated action (cases 2.1.2.2. and 2.2.2. and $l > 0$) we have $F_i(\alpha) = \frac{N}{N-(l+1)} \times \epsilon$.

Therefore we have $F_i(\alpha) < 1$ iff $\epsilon < \frac{N-(l+1)}{N}$ (therefore we also have $\epsilon < \frac{N-1}{N}$), and we have $F_i(\alpha) > 1$ iff $\epsilon > \frac{N-(l+1)}{N}$ (and this is consistent with $\epsilon < \frac{N-1}{N}$).

Therefore, according to principle D we may have either $F_i(\alpha) < 1$ or $F_i(\alpha) > 1$ depending on the values of ϵ , l and N.

[7] Notice that the case $N - (l + 1) = 0$ can be rejected because if $l = N - 1$ there is only one restricted tolerated action and the agent has no choice offered by Σ_k.

4.4 Coming back to the example

The method we have presented can be used to compute **iteratively** the values of $P(Int(\alpha, s_i)|O_{1,i})$, $P(Int(\beta, s_i)|O_{1,i})$ and $P(Int(\gamma, s_i)|O_{1,i})$.

If we use the notations $\Pi_i(\alpha) = P(Int(\alpha, s_i)|O_{1,i})$, $\Pi_i(\beta) = P(Int(\beta, s_i)|O_{1,i})$, $\Pi_i(\gamma) = P(Int(\gamma, s_i)|O_{1,i})$, and $R = N \times (1 - \epsilon)$, for recommended actions, and $T = \frac{N}{N-1} \times \epsilon$ for tolerated actions we get the following table

s_i	$\Pi_i(\alpha)$	$\Pi_i(\beta)$	$\Pi_i(\gamma)$
s_0	$\pi(\alpha)$	$\pi(\beta)$	$\pi(\gamma)$
s_1	$\pi(\alpha)$	$\pi(\beta)$	$\pi(\gamma)$
s_2	$N \times \pi(\alpha)$	$\pi(\beta)$	$N \times \pi(\gamma)$
s_3	$N \times T \times \pi(\alpha)$	$N \times \pi(\beta)$	$N \times T \times \pi(\gamma)$
s_4	$N \times R \times T \times \pi(\alpha)$	$N \times T \times \pi(\beta)$	$N \times T^2 \times \pi(\gamma)$
s_5	$\pi(\alpha)$	$N \times T^2 \times \pi(\beta)$	$N \times T^3 \times \pi(\gamma)$

We have $N > R > 1$ and $T < 1$.

If we have $\pi(\alpha) = \pi(\beta) = \pi(\gamma)$ we can determine what the system believes about the agents' intentions in these situations. As expected in 4.1 we get:

In s_0 we have $BInt(\alpha, s_0)$, $BInt(\beta, s_0)$ and $BInt(\gamma, s_0)$.

In s_1 we have $BInt(\alpha, s_1)$, $BInt(\beta, s_1)$ and $BInt(\gamma, s_1)$.

In s_2 we have $BInt(\alpha, s_2)$ and $BInt(\gamma, s_2)$.

In s_3 we have $BInt(\beta, s_3)$.

In s_4 we have $BInt(\alpha, s_4)$.

In s_5 we have $BInt(\beta, s_5)$.

5 Conclusion

We have presented a method to assign intentions to an agent which is based on the computation of the estimation of the probability that an agent has the intention to perform a procedure.

There are two parts in the computation method. The first part (section 4.2) is general and is based on the assumptions H1-H4. The second part (section 4.3) is based on heuristics and on the additional assumptions H5-H7 and requires to know the value of $\pi(\alpha)$ for each α. The values of N and l are determined by the application domain and the value of ϵ can be tuned by a designer.

A difference with other methods for plan recognition is that in the procedures we may have terms of the form σ/β. The property *Doing* allows the selection of the procedure that matches the observations $O_{1,i}$. To estimate the probability of the occurrence of the next observation O_i we consider the part α'_{i-1} of the procedure α that has already been performed. Therefore the estimated probabilities depend on the history and not just on the previous observation O_{i-1}. This is an important original aspect of the method.

The computation cost of the estimated probabilities and of the evaluation of the properties *Doing* and *Done* is linear with respect to the number of observations for a given procedure. That makes the computation very fast.

Finally, it is worth noting that a preliminary version of the method has been implemented in Prolog [10]. This implementation was of great help to check our intuition on simple examples.

Future works will be:
1) to remove the too strong assumption H6 about the independence of the observations O_i in order to have a better estimation of $\frac{P(O_i|O_{1,i-1} \wedge Int(\alpha, s_{i-1}))}{P(O_i|O_{1,i-1})}$,
2) to guarantee that after a long sequence of observations of tolerated actions $P(Int(\alpha, s_i)|O_{1,i})$ is never lower than $\pi(\alpha)$ and
3) to allow test actions ϕ? and temporal conditions in the procedure definitions.

Acknowledgment. We are very grateful to G. Eizenberg for his help in Probability Theory. If there are some errors they are the own responsibility of the authors.

References

1. D. W. Albrecht, I. Zukerman, and A. E. Nicholson. Bayesian models for keyhole plan recognition in an adventure game. In *User modeling and user-adapted interaction*, volume 8, pages 5–47. 1998.
2. D. E. Appelt and M. E. Pollack. Weighted abduction for plan ascription. In *User modeling and user-adapted interaction*, volume 2, pages 1–25. 1991.
3. J. A. Baier. On procedure recognition in the Situation Calculus. In *22nd International Conference of the Chilean Computer Science Society*. IEEE Computer Society, 2002.
4. M. Bauer. Integrating probabilistic reasoning into plan recognition. In *Proceedings of the 11th European Conference of Artificial Intelligence*. John Wiley and Sons, 1994.
5. N. Blaylock and J. Allen. Corpus-based, statistical goal recognition. In G. Gottlob and T. Walsh, editors, *Proceedings of the 18th Inernational Joint Conference on Artificial Intelligence*, pages 1303–1308, 2003.
6. S. Carberry. Incorporating default inferences into plan recognition. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 471–478. 1990.
7. E. Charniak and R. P. Goldman. A Bayesian model of plan recognition. *Artificial Intelligence*, 64(1), 1993.
8. P. R. Cohen and H. J. Levesque. Persistence, Intention, and Commitment. In A. L. Lansky M. P. Georgeff, editor, *Reasoning about actions and plans*, pages 297–340, Timberline, USA, 1986.
9. R. Demolombe and E. Hamon. What does it mean that an agent is performing a typical procedure? A formal definition in the Situation Calculus. In C. Castelfranci and W. Lewis Johnson, editor, *First International Joint Conference on Autonomous Agents and Multiagent Systems*. ACM Press, 2002.
10. A. M. Otermin Fernndez. Reconocimiento de intenciones de un operador que interacta con un sistema. Technical Report, ONERA Toulouse, 2004.
11. H. A. Kautz. A formal theory of plan recognition and its implementation. In J. F. Allen, H. A. Kautz, R. N. Pelavin, and J. D. Tennemberg, editors, *Reasoning about plans*, pages 69–126. Morgan Kaufman Publishers, 1991.
12. H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. GOLOG: A Logic Programming Language for Dynamic Domains. *Journal of Logic Programming*, 31:59–84, 1997.

13. W. Mao and J. Gratch. A utility-based approach to intention recognition. In *Proceedings of the AAMAS 2004 Workshop on Agent Tracking: modeling other agents from observations*, 2004.
14. P. Pozos Parra, A. Nayak, and R. Demolombe. Theories of intentions in the framework of Situation Calculus. In *Proceedings of the AAMAS workshop on Declarative Agent Languages and technologies*, 2004.
15. R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* MIT Press, 2001.

Ability in a multi-agent context: a model in the Situation Calculus

Laurence Cholvy[1], Christophe Garion[2], and Claire Saurel[1]

[1] ONERA Centre de Toulouse, 2 av Édouard Belin, 31055 Toulouse, France
[2] SUPAERO, 10 av Édouard Belin, 31055 Toulouse

Abstract. This paper studies the notion of ability and its relation with the notion of action in a multi-agent context. It introduces the distinction between two notions respectively called "theoretical ability" and "ability". The main contribution of this paper is a model of these notions in the Situation Calculus.

1 Introduction

Allocating tasks or planning in a multi-agent context [8], [3], requires taking into account what the agents are able to do, i.e., the agent abilities, in order to assign tasks to agents who are able to perform them.

The notion of ability must then be modelled and this implies to explicit the parameters which define this notion.

Obviously, the agent itself, or the group of agents, is one of these parameters. But what is the nature of what the ability applies on? For instance, when we say that *John is able to paint the door*, do we mean that John is able to perform a particular action which consists in applying paint with a specific brush on the door? Or do we mean that John is able to see to it that the door is painted, by the means he wants, for instance by delegating this task to someone else? Modelling ability thus implies modelling actions.

In the literature, there are two main approaches to action theory. The first one consists in giving in the language means to explicitly represent actions. This is the case of dynamic logic for instance [9], which offers modal operators to speak about the execution of an action, and also the execution of an action by an agent. This is also the case of situation calculus [14, 16], which allows one to represent actions, their preconditions and their effects, but also situations, considered as results of the successive application of actions in an initial situation. On the contrary, actions are not explicitly represented in the second approach. The operators defined there only allow to express the fact that the agent sees about some property to be true (cf. the operator *stit* [11] and the notion of *agency* in [7]).

As the notion of ability is strongly linked to the notion of action, it has been studied according to these two approaches. For instance, the multi modal dynamic logic KARO [18] aims at defining agent ability to perform an action according to the first approach. Primitive concepts are the agent's knowledge, its capacity to perform an action, the effects of an action and the opportunity associated with an action. Ability and opportunity are two intertwined notions, we will come back on it later.

Concerning the second approach, the notion of ability does not bear on actions, but on the fact that a property is true [10, 7]. These two formalisms are based on propositional modal logics. In [7], Elgesem defines ability and action as primitive notions. He considers a function f which determines for a given world w and a goal φ the worlds in which the agent has realized its ability to see to it that φ is true from w. Thus, an agent is able to see to it that φ is true if and only if the set of worlds $f(w, \varphi)$ is not empty. With this definition, ability and action, which is also defined by f, are two binded notions. For instance, if an agent sees to it that φ is true, then this agent is able to see to it that φ. In [10], Horty uses temporal models to represent actions: an agent sees to it that φ is true at a moment m if it restricts the "histories" which m belongs to in order that φ is true. The ability for an agent to see to it that φ is true is defined as the possibility (in the classical sense, see [2]) for the agent to see to it that φ is true. Let us notice that with this formalism, Horty avoids several paradoxes. In particular, it cannot be deduced that if φ is true, then the agent has the ability to see to it that φ.

We must also mention [12], in which the authors use the situation calculus to model the notion of "ability to reach a goal", i.e., "ability to make a proposition true". Two definitions are given in a mono-agent context. According to the one the authors find the simplest to use, the agent has the ability, in a situation s, to make φ true (i.e., the agent is able to reach the goal φ) if there exists a sequence of actions such that the agent knows in s that executing these actions will make φ true. In other terms, the agent has the ability to make φ true if he knows a plan to achieve φ.

This brief state of the art shows that there is no consensus on what the ability applies on. However, we find in the literature several points of agreement relatively to the notion of ability.

First, the notion of ability must not be confused with the notion of possibility nor with the notion of permission [17]. These possible confusions are due to ambiguities of the natural language. For instance, the sentence "I can open the door" is sometimes used to say "I am able to open the door" according to the notion of ability we study here. But this sentence is also sometimes used to say "I have now the possibility to open the door" (because, for instance, the door is now unlocked), but this does not mean that I am able to do so. Here, it refers to a notion of possibility. Finally, this sentence is sometimes used to mean "I have the permission to open the door", which still does not imply that I am able to do so and which refers here to a deontic notion.

Secondly, several people agree on the fact that two kinds of ability must be distinguished [17, 18, 5, 1].

One can first distinguish what is called "generic ability" by some people (or "ability" by others) and which refers to the agent's competences to perform an action in normal conditions. "I can open the door" means here that I know what to do to open the door, independently of my current intellectual or physical state and of the current state of the world. Thus, with this point of view, I can say "I am able to open the door" even if my arms are broken or the door is locked.

One can also distinguish what is called "occasional ability" by some people (or "pragmatic possibility", or "opportunity to exercise an ability" by others) and which refers to the current situation. Here, "I can open the door" means that I have the generic ability to open the door *and* the current situation is such that conditions are favourable for me to use this generic ability (for instance, my arms are not broken and the door is unlocked).

Finally, in a multi-agent context, one of the main problems is to define the notion of ability relatively to a group of agents, and in particular to infer the ability of the group from the abilities of the individuals of the group. For instance, what are the conditions for saying that a group of people is able to paint the door? Or a group of people is able to first sand the door, then to paint it? The problem is not trivial since the notion of ability previously called "occasional" is a dynamic notion which depends on the state of the world in which it is evaluated. But in a multi-agent context, the dynamic of the world is hard to foresee because several agents may change the world.

This paper presents a preliminary study of the notion of ability in a multi-agent context. As far as we know, no previous work has already attacked the same question in a multi-agent context. In particular, it must be noticed that the notion we are trying to model here is different from the one Pauly has studied [15]. Indeed, in his work "an agent (or a group of agents) can bring about a proposition" means that this agent (or group of agents) has a (collective) efficient strategy which makes this proposition true, whatever the other people do. In particular, the logic defined by Pauly does not apply when a first agent can bring about a proposition and another one can bring about its contrary.

This paper is organized as follows. In section 2, we informally discuss some requirements about the notion of ability. In section 3 we propose a formal model, in the Situation Calculus, of concepts related to ability and we justify the use of such formalism. Some properties of this model are given in 4, and an example is detailed in 5. Section 6 presents an implementation of our modelling and section 7 is devoted to a discussion and presents some perspectives.

2 Informal requirements about ability

Our modelling lies on the following choices:

- The ability we focus on bears on actions. We aim at characterizing the meaning of being able to perform actions, i.e., to perform a procedure [6].
- We aim at explicitly representing actions, their preconditions and their effects. We consider the general case when some primitive actions may require several agents to be performed (for instance, lifting an heavy door requires two agents).
- We aim at defining the ability of an agent to perform an action as the combination of its competences and some favourable conditions that allow it

to perform that action. But we also aim at making a difference between the conditions which are related to the agent only from those which are not. For doing so, we introduce an intermediary notion called "theoretical ability".

2.1 Model of action

In our model of action, any primitive action is specified by its preconditions which are the conditions under which it can be performed, independently of any agent. When these conditions are true, we say that *the action is possible* [16].

Example 1. For instance, "painting" is only possible if there is some paint and a brush.

2.2 Model of ability

In our model, the primitive notion is the notion of competence described as follows:

Competence Competence represents the knowing-how of the agent (or agents) relatively to an action. This knowledge may be inborn or may result from a learning phase. In our model, this information is considered as *primitive*.

For instance, we will have initial data like: "John is competent to paint a door" or "John and Peter are competent together to lift the door". The first sentence means that John knows the successive gestures he has to make so that the door is painted. The second sentence means that they both know how to coordinate their gestures in order to lift the door.

In this work, we will assume that the competences of the agents can not be deleted: once an agent is competent to perform an action, he will always be. This assumption seems to be justified in many applications with rather short temporal horizon where it can be assumed that the agents do not loose their competences. As we will see, this assumption can be easily removed.

It must be noticed that this notion of competence is different from the one Cohen and Levesque consider in [4], where, if an agent competent for a proposition p believes p, then p is true.

Theoretical ability From the notion of competence,we first define the notion of theoretical ability as follows:

Definition 1. *Let A be a non empty group of agents (possibly a singleton) and α be a primitive action. A is theoretically able to perform α if:*

1. *A is competent to perform α*
2. *some conditions, related to the agents of A, are true.*

Remember that competence is considered to be a primitive notion. The conditions expressed in point 2 concern the agent (its physical state for instance), but not all the environment.

Example 2. For instance, an agent is theoretically able to paint a door if it is competent for paint a door and if it is not tired.

Notice that this notion is close to the notion of *ability* of [18].

Ability The notion of ability is finally defined from the notion of theoretical ability by taking into account the conditions which define the possibility to perform the action.

Definition 2. *Let A be a non empty group of agents (possibly a singleton) and α be a primitive action. A is able to perform α if:*

1. *A is theoretically able to perform α*
2. *α is possible.*

Notice that the notion of possibility in this definition is the one defined by Reiter in [16]. The possibility here is a set of conditions concerning the state of the world excepting the agent.

Example 3. For instance, an agent is able to paint a door if it is theoretically able to paint the door (i.e, competent for painting the door, not tired) and if there is some paint and a brush.

This notion of ability is a kind of occasional ability in the sense of section 1.

It can also be noticed that the previous condition "α is possible" is very close to the notion of "opportunity to exercise an ability" mentioned in [5], as well as the one mentioned in [18]. It means that, in our approach, an agent is able to perform an action if it has the opportunity to exercise the theoretical ability to perform this action.

2.3 Extensions to more complex actions

Considering only primitive actions is not enough and we must also consider more complex actions obtained by composition of primitive ones. In this preliminary work, we focus on sequences (like for instance: "lift the door, then paint it".)

We would like to validate the following assertion (cf. [18]): agent a is able to perform the sequence α then β if a is able to perform α and, once a has performed α, a is able to perform β.

Example 4. For instance, assume that sanding a door is tiring. Then, we would like to deduce that John is not able to sand then to paint the door (i.e, not able to perform the action "sand then paint"). Indeed, even if John is able to sand the door, once he will have sanded it, he will be tired. Thus, he will not be able to paint the door.

2.4 Deriving ability of a group from abilities of individuals

In a multi-agent context, it is necessary to extend these previous notions for a group of agents. We will say that a group of agents is able to perform a primitive action if one of its sub-group (possibly a singleton) is able to perform it. Notice that in some cases, only a subgroup may be competent to do some action: an agent does not have the competence to carry a piano, but a group of three agents may have it.

As for the sequences, we would like to validate the fact that a group is able to perform the action "α then β" if one of its sub-group is able to perform α and, once this sub-group has performed α, the group is able to perform β.

Example 5. Consider now that Peter is also competent to paint the door and is not tired. Then the group {John, Peter} is able to sand the door then to paint it. Indeed, John is able to sand it (see previously) and, once John has sanded the door, the group is still able to paint the door (because Peter has remained not tired).

3 Model of ability in the Situation Calculus

3.1 The Situation Calculus

We suggest to use the Situation Calculus to model these notions for two reasons:

- firstly, this formalism is a good candidate for modelling actions since it offers means to explicitly express preconditions and effects of actions;
- secondly, an important problem underlying this present work, the frame problem (i.e, how to express what are the changings induced by the performance of an action by an agent and how to express what remains unchanged), has been provided a solution in the Situation Calculus by Reiter.

3.2 The language

We consider a first order language $\mathcal{L}_{CS}$ which will allow us to model and reason about actions and ability. In this language, the changes of the world are resulting from action performances. It is defined as follows:

- a set of constants to represent agents A.
- a set of functions and constants used to represent primitive actions, with parameters or without.
  For instance the term "paint(x)" will represent the action "to paint the object x".
- A unary predicate $primitive(.)$ used to list the primitive actions.
  Thus, $primitive(paint(x))$ means that $paint(x)$ is a primitive action.
- a binary function ; used to represent the sequence of actions.
  $sand(x); paint(x)$ will represent the action which consists in sanding the object x then painting it.

– a constant S_0 used to represent the initial situation.
– a ternary function *do*.
  $do(\mathcal{A}, paint(x), s)$ represents the situation which follows from the situation s, when the group of agents $\mathcal{A}$ has painted the object x.
  Notice that here, unlike the "classical" Situation Calculus, the agent is not a parameter of the function which represents the action, but is a parameter of the function *do* which represents the performance of the action.
– a set of predicates called relational *fluents* which represent properties which may be changed by the performance of an action. The last argument of a fluent is a situation.
  For instance, $painted(door, S_0)$ expresses that the door is painted in the initial situation S_0.
– a particular binary *fluent Poss* used to express that an action is possible in a situation.
– a particular binary *fluent* is *competent* and is used to represent the fact that an agent (or a group) is competent for performing a primitive action.
  For instance, $competent(\{a\}, paint(door), s)$ expresses that agent a is competent to paint the door in situation s.
– a particular ternary *fluent* is *able_t* and is used to represent the fact that an agent (or a group) is theoretically able to perform an action.
  $able\_t(\{a\}, paint(door), S_0)$ expresses that agent a is, in situation S_0, theoretically able to paint the door.
– a particular ternary *fluent* is *able* and is used to represent the fact that an agent (or a group) is able to perform an action.
  $able(\{a\}, paint(door), S_0)$ expresses that agent a is, in situation S_0, able to paint the door.

3.3 The axioms

Description of the initial situation First, the initial state of the world must be represented. For doing so, for any fluent f and for any tuples $t_1, \ldots, t_n$ of ground terms such that $f(t_1, \ldots, t_n)$ is true in the initial situation, we consider the following axiom:

$$f(t_1, \ldots, t_n, S_0) \tag{1}$$

In particular, since *competent* is a fluent, for any group G competent for performing the primitive action α in the initial situation S_0, we consider the following axiom:

$$competent(G, \alpha, S_0) \tag{2}$$

Primitive actions For any primitive action α, we consider an axiom of the following form:

$$primitive(\alpha) \tag{3}$$

Precondition axioms for primitive actions We represent the preconditions of the primitive actions (i.e., the conditions that make the performance of the action possible) by an axiom of the following type:

$$\forall\alpha\forall S\ Poss(\alpha, S) \leftrightarrow pre(\alpha, S) \tag{4}$$

Precondition axioms for sequence We then extend this kind of axioms for a sequence $\alpha;\beta$ where α is a primitive action and β is a complex action as follows:

$$\forall S\forall G\forall\alpha\forall\beta\ \ Poss(\alpha, S) \wedge Poss(\beta, do(G, \alpha, S)) \leftrightarrow Poss(\alpha;\beta, S) \tag{5}$$

Axiom (5) expresses that $\alpha;\beta$ is possible in S iff α is possible in S and β is possible after the performance of α in S.

Successor state axioms Following Reiter [16], for any *fluent* $f(t_1, \ldots, t_n)$, we consider a successor state axiom which specifies all the ways the value of the fluent may change.

$$\forall S\forall G\forall\alpha\ \ Poss(\alpha, S) \rightarrow f(t_1, \ldots, t_n, do(G, \alpha, S)) \leftrightarrow \tag{6}$$
$$\gamma_f^+(t_1, \ldots, t_n, \alpha, S) \vee (f(t_1, \ldots, t_n, S) \wedge \neg\gamma_f^-(t_1, \ldots, t_n, \alpha, S))$$

$\gamma_f^+(t_1, \ldots, t_n, \alpha, S)$ represents the conditions which make f true after α has been performed in S. $\gamma_f^-(t_1, \ldots, t_n, \alpha, S)$ represents the conditions which make f false after α has been performed in S.

Since *competent* is a *fluent*, we have to express a successor state axiom for it. In this paper, we assume that the competence is not deleted, i.e once an agent is competent to perform an action. This is expressed by:

$$\forall\alpha\forall\beta\forall G\forall S\ \ Poss(\beta, S) \rightarrow (competent(G, \alpha, do(G, \beta, S)) \leftrightarrow competent(G, \alpha, S)) \tag{7}$$

One can wonder why we have chosen to use a fluent to represent competence if we assume that competence does not change during execution of action. Let us claim that our modelling allows to relax this assumption easily by modifying axiom (7).

Theoretical ability axioms For any primitive action α, we consider an axiom of the following form:

$$\forall G\forall S\ \ competent(G, \alpha, S) \wedge conditions\_t(G, \alpha, S) \rightarrow able\_t(G, \alpha, S) \tag{8}$$

It expresses that a group G is theoretically able to perform α in situation S if G is competent for α in S and if some conditions related to G and α are satisfied.

Finally, in order to derive the theoretical ability for a group of agents, we consider:

$$\forall G \forall G' \forall \alpha \forall S \;\; primitive(\alpha) \wedge (G' \subseteq G) \wedge able\_t(G', \alpha, S) \rightarrow able\_t(G, \alpha, S) \quad (9)$$

$$\forall G\, G'\, \alpha\, \beta\, S \;\; (G' \subseteq G) \wedge able\_t(G', \alpha, S) \wedge able\_t(G, \beta, do(G', \alpha, S)) \rightarrow able\_t(G, \alpha; \beta, S) \quad (10)$$

Axiom (9) expresses the fact that if a sub group G' of G is theoretically able to perform a primitive action α, then the group G is also theoretically able to perform α. Axiom (10) expresses that if a sub-group G' of G is theoretically able to perform α and if G is theoretically able to perform β once G' has performed α, then G is theoretically able to perform $\alpha; \beta$ (i.e., to perform α then β).

Ability axioms Finally, the following axiom allows to derive the ability of a group:

$$\forall G \forall \alpha \forall S \; able\_t(G, \alpha, S) \wedge Poss(\alpha, S) \rightarrow able(G, \alpha, S) \quad (11)$$

4 Some properties of this model

Proposition 1. *Let $\Sigma = \{(1), \ldots, (11)\}$ be the set of axioms presented previously. Then :*

$$\Sigma \vdash \forall \alpha \forall \beta \forall G \forall G' \forall S \;\; able(G, \alpha, S) \wedge able(G', \beta, do(G, \alpha, S)) \rightarrow able(G \cup G', \alpha; \beta, S)$$

This proposition is proved by an inductive proof on the length of the sequence $\alpha; \beta$.

This proposition means that if the group G is able to perform α in the situation s and if the group G' is able to perform β after G has performed α in S, then the group $G \cup G'$ is able to perform $\alpha; \beta$ in S.

A corollary is the following:

Proposition 2. *Let $\Sigma = \{(1), \ldots, (11)\}$ be the set of axioms presented previously. Then :*

$$\Sigma \vdash \forall \alpha \forall \beta \forall G \forall S \;\; able(G, \alpha, S) \wedge able(G, \beta, do(G, \alpha, S)) \rightarrow able(G, \alpha; \beta, S)$$

Proposition 3. *Let $\Sigma = \{(1), \ldots, (11)\}$ be the set of axioms presented previously, f be a fluent and α a primitive action.*
$\Sigma \vdash \forall G \forall S \;\; Poss(\alpha, S) \rightarrow f(\ldots, do(G, \alpha, S)) \not\Longrightarrow \Sigma \vdash \forall S \forall G' \;\; f(\ldots, S) \rightarrow able(G', \alpha, S)$.

This proposition is proved by finding a counter example.

This result guaranties that we do not validate the paradox mentioned in introduction: "if φ is true, then the agent has the ability to see to it that φ". Reformulated in our model, this come to say that we do not validate: if α is an action so that a postcondition is that f is true (i.e, after any performance of α by a group of agents G, f is true) and if f is true in a situation S, then any group of agents G' is able to perform α.

5 Example

5.1 Description of the example

Let a, b and c be three agents.

- Primitive actions we consider are: to lift the door ($lift$), to sand the door ($sand$), to paint the door ($paint$).
- Competence of agents are: a is competent for sanding the door and for painting it. b is only competent for painting the door. a and b together are competent for lifting the door. c is not competent for any action.
- The initial situation is such that there is a sander ($sander(S_0)$) and it works, there is some paint and the agents are not tired (for each agent a, $ok(a, S_0)$ holds).
- Sanding is possible if the sander works.
- Painting is possible if there is some paint ($paint_r$).
- An agent is theoretically able to sand the door if it is competent for doing so and if it is not tired ; an agent is theoretically able to paint the door is it is competent for doing so and if it is not tired ; two agents are together theoretically able to lift the door is they are together competent for doing so and if they are not tired.
- Successive state axioms are defined as follows:
  An agent is tired iff it has sanded the door, or it has participated in lifting the door.
  There is paint left after the execution of an action, except if it is a painting action.
  No action makes the sander out.

5.2 Formulas in the Situation Calculus

Description of the initial situation (axioms (1) and (2))

$$ok(a, S_0)$$
$$ok(b, S_0)$$
$$ok(c, S_0)$$
$$paint\_r(S_0)$$
$$sander(S_0)$$
$$competent(\{a, b\}, lift, S_0)$$

$$competent(\{b,a\}, lift, S_0)$$
$$competent(a, sand, S_0)$$
$$competent(a, paint, S_0)$$
$$competent(b, paint, S_0)$$

Primitive actions (axioms (3))

$$primitive(lift)$$
$$primitive(sand)$$
$$primitive(paint)$$

Preconditions axioms for primitive actions (axioms (4))

$$\forall S \;\; Poss(lift, S)$$

$$\forall S \;\; sander(S) \leftrightarrow Poss(sand, S)$$

$$\forall S \;\; paint\_r(S) \leftrightarrow Poss(paint, S)$$

Successive state axioms (axioms (6))

$$\forall B \forall Y \forall S \;\; Poss(Y, S) \rightarrow (sander(do(B, Y, S)) \leftrightarrow sander(S))$$

$$\forall A \forall X \forall S \;\; Poss(X, S) \rightarrow (paint\_r(do(A, X, S)) \leftrightarrow paint\_r(S) \wedge \neg(X = paint))$$

$$\begin{gathered} \forall A \forall B \forall X \forall S \;\; poss(X, S) \rightarrow (ok(A, do(B, X, S)) \leftrightarrow \\ ((ok(A, S) \wedge (B = A) \wedge \neg(X = sand)) \vee \\ (ok(A, S) \wedge (A \in B) \wedge \neg(X = lift)) \vee \\ (ok(A, S) \wedge \neg(A \in B)))) \end{gathered}$$

Theoretical ability axioms (axioms (8))

$$\forall A \forall B \forall S \;\; competent(A, B, lift, S) \wedge ok(A, S) \wedge ok(B, S) \rightarrow able\_t(\{A, B\}, lift, S)$$

$$\forall A \forall S \;\; competent(A, sand, S) \wedge ok(A, S) \rightarrow able\_t(A, sand, S)$$

$$\forall A \forall S \;\; competent(A, paint, S) \wedge ok(A, S) \rightarrow able\_t(A, paint, S)$$

5.3 Some conclusions

Let us denote Σ the set of axioms (1),...,(11). Then,

- $\vdash \Sigma \rightarrow able\_t(a, paint, S_0)$
  In the initial situation, a is theoretically able to paint the door because it is competent for doing it and it is not tired.
- $\vdash \Sigma \rightarrow able\_t(a, paint, do(a, paint, S_0))$
  Since painting does not make the agent tired, a is still theoretically able to paint after he has painted.
- $\nvdash \Sigma \rightarrow able(a, paint, do(a, paint, S_0))$
  But, after a has painted the door, there is no more paint, so it cannot be proved that a is able to paint the door again (even if it is theoretically able as it is shown previously)
- $\nvdash \Sigma \rightarrow able\_t(a, sand; paint, S_0)$
  Indeed, after having sanded the door, a will be tired, so he will not be theoretically able to paint the door.
- $\nvdash \Sigma \rightarrow able\_t(a, paint, do(\{a, b\}, lift, S_0))$
  After the group a, b has lifted the door, a and b are tired. Thus, a is not theoretically able to paint the door.
- $\vdash \Sigma \rightarrow able(\{a, b\}, paint; lift, S_0)$
  The group a, b is able to paint then lift the door. Indeed once a or b will have painted the door, a and b will not be tired. So they will be able to lift the door.
- $\nvdash \Sigma \rightarrow able(\{a, b\}, lift; paint, S_0)$
  The group a, b is not able to lift then to paint the door. Indeed once a and b will have lifted the door, a and b will both be tired. So none of them will be able to paint the door.
- Let us add now that agent c is competent for painting the door:
  Let Σ' the set obtained by adding the formula $competent(c, paint, S_0)$ to Σ.
  Thus $\vdash \Sigma' \rightarrow able(\{a, b, c\}, lift; paint, S_0)$
  The group a, b, c is now able to lift then to paint the door. Indeed, if agents a and b lift the door, then this does not make c tired. So c is able to paint the door after a and b have lifted the door.

6 Implementation

This model has been implemented in Prolog. As in [13], we use a binary predicate `holds` in order to represent fluents. For instance, $ok(a, S_0)$ is represented by `holds(ok([a]), S0)`. The successor state axiom for fluent ok is expressed by the following clause:

```
holds(ok(A), do(B,X,S)) :-
   B=A, \+ (X=sand), holds(ok(A),S), Poss(X,S);
   member(A,B), \+ (X=lift), holds(ok(A),S), Poss(X,S);
   \+ member(A,B), holds(ok(A),S), Poss(X,S).
```

We can show that, given a group of agents G and an action α primitive or not, we have $\Sigma \vdash able\_t(G, \alpha, S_0)$ (resp. $\Sigma \vdash able(G, \alpha, S_0)$) if and only if Prolog with negation as failure proves `holds(able_t(G,alpha),S0)` (resp. `holds(able(G,alpha),S0)`). For instance, resuming the example presented in section 5.3 using negation as failure, we can now show that the answer to the question `holds(able([a,b], [paint,lift], S0))` is `yes` and the answer to `holds(able([a,b], [lift,paint], S0)` is `no`[3].

7 Discussion

In this paper, we have presented an attempt to model in the Situation Calculus the notions of theoretical ability and ability of an agent towards an action in a multi-agent context. Definitions of these two notions to groups of agents has also been given.

In this model, agents' theoretical ability depends on their competence and on some conditions depending on the agents. Agents' ability is then defined from theoretical ability and from some conditions which do not depend on the agents.

Introducing the notion of theoretical ability is of course interesting from a modelling point of view since it makes a distinction between conditions which are related to the agents who perform the actions and conditions which are not. But it may also be interesting in the preliminary phase of planning when choosing the agents who will be in charge of the task to be performed. Indeed, proving that the chosen agents are not even theoretically able to perform the global task is enough to prove that the task will never be performed by these agents and that changing agents is required.

However, if one is only interested in proving that a group of agents is able to perform an action, the intermediary notion of theoretical ability is not useful and definitions have to be compacted as follows:

$$\forall G \forall S \quad competent(G, \alpha, S) \land conditions\_t(G, \alpha, S) \land Poss(\alpha, S) \to able(G, \alpha, S)$$

This preliminary work has many perspectives.

First, some more formal properties on this model must be proved. In particular, formal relations with existing works mentioned in the introduction have to be established.

Secondly, another assumption could be made when inferring the ability of a group from the abilities of its agents. Indeed, the model presented here assumes that a group of agents is able to perform an action if one of its member is able to do so. But this assumes that the conditions for an agent to be theoretically able to perform an action do not depend on the fact that this agent belongs or not to a group. But it could happen that a single agent is theoretically able to perform an action but when it belongs to a group, it is no longer able (not because the others agents prevent him to do so but because belonging to a group changes the conditions sufficient for him to be theoretically able to perform the action).

[3] Notice that we implement sequence of actions as lists in Prolog.

Thirdly, we have to extend this work by considering more types of complex actions like concurrence, iteration or conditionals. We must also take into account time and action durations. For doing so, the solution provided in [6] can be adopted.

Finally, the model presented here does not take external actions into account. In particular, fluents are changed only by actions performed by the agents we consider. But in many applications, the world may change because some other agents we don't know change it. A immediate solution we could study, consists in introducing an "external agent" who could be used to model the evolution of the world which are independent from the other agents.

Acknowledgements

This work has been funded by DGA (Délégation Générale de l'Armement) under contract: SPOTI/0373088.

References

1. B. Chaib-Draa and R. Demolombe. L'interaction comme champ de recherche. *Information - Interaction - Intelligence, numéro spécial Modèles Formels de l'Interaction*, pages 5–24, 2001.
2. B.F. Chellas. *Modal logic. An introduction.* Cambridge University Press, 1980.
3. L. Cholvy and C. Garion. Distribution of goals addressed to a group of agents. In J. S. Rosenschein, T. Sandholm, M. Wooldridge, and M. Yokoo, editors, *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 765–772. ACM Press, July 2003.
4. P.R. Cohen and H.J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.
5. R. Demolombe. Formalisation en logique des interactions entre agents : quels concepts formaliser ? Technical report, ONERA/DTIM, 2000. In French.
6. R. Demolombe and E. Hamon. What does it mean that an agent is performing a typical procedure: a formal definition in the Situation Calculus. In *Proceedings of the First International Joint Conference an Autonomous Agents and Multiagent Systems (AAMAS'02)*, pages 905–911, Bologne, 2002.
7. D. Elgesem. The modal logic of agency. *Nordic Journal of Philosophical Logic*, 2(2):1–46, 1997.
8. B.J. Grosz, L. Hunsberger, and S. Kraus. Planning and acting together. *AI Magazine*, 20(4):23–34, 1999.
9. D. Harel. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume 2, pages 497–604. D. Reidel Publishing Company, 1984.
10. J.F. Horty. Agency and obligation. *Synthese*, 108:269–307, 1996.
11. J.F. Horty and N. Belnap. The deliberative stit : a study of action, omission, ability and obligation. *Journal of Philosophical Logic*, 24:583–644, 1995. Reprinted in *The Philosopher's Annual, Volume 18-1995*, Ridgeview Publishing Company, 1997.
12. Y. Lespérance, H.J. Levesque, F. Lin, and R.B. Scherl. Ability and knowing how in the situation calculus. *Studia Logica*, 66(1):165–186, oct 2000.
13. H.J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–84, 1997.

14. J. MacCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In D. Michie and B. Melzer, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.
15. M. Pauly. A modal logic for conditional power in games. *Journal of Logic and Computation*, 12(1):149–166, 2000.
16. R. Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, New York, 1991.
17. R. Thomason. Ability, action and context. Presentation at the Temporality and Discourse Context: Dynamic and Modal Approaches Workshop, 2001.
18. B. van Linder, W. van der Hoek, and J.-J. Ch. Meyer. Formalizing abilities and opportunities of agents. *Fundamenta Informaticae*, 34(1-2):53–101, 1998.

Trustworthiness by default

Johan W. Klüwer[1] and Arild Waaler[2]

[1] Dep. of Philosophy, University of Oslo johanw@filosofi.uio.no
[2] Finnmark College and Dep. of Informatics, University of Oslo, arild@ifi.uio.no.

But never put a person to death on the testimony of only one witness. There must always be at least two or three witnesses.
Deuteronomy 17:6 (New Living Translation)

Abstract. We present a framework for reasoning about trustworthiness, with application to conflict resolution and belief formation at various degrees of reliability. On the basis of an assignment of relative trustworthiness to sets of information sources, a lattice of degrees of trustworthiness is constructed; from this, a priority structure is derived and applied to the problem of forming the right opinion in the presence of possibly conflicting information. Consolidated with an unquestioned knowledge base, this provides an unambiguous account of what an agent should believe, conditionally on which information sources are trusted. Applications in multi-agent doxastic logic are sketched.

1 Introduction

To trust an information source, in the simplest, unconditional form, is to believe every piece of information that the source provides. While providing a paradigm, this notion of trust has limited application to realistic scenarios. In general, the trust we have in our information sources, which may vary in kind from teachers to newspapers to legal witnesses, is not unconditional: we believe what we are told by a trusted source only as long as we don't possess knowledge to the contrary. This simple observation motivates the approach to trust that we will be discussing in this paper. Conditional trust in an information source is a *default* attitude: To believe what you are told, unless you know better.

When looking for information, we often need to consider several sources. Sources may vary widely with regard to their reliability, and a cautious default approach then informs us to let the more trustworthy ones take priority over those that are less trustworthy. Furthermore, we often need to consider more than one source at a time. Notions of agreement or corroboration, as well as the consolidation of information drawn from different sources, are essential.

What we present here is a framework for reasoning about relative trustworthiness, with *sets* of information sources as the basic trusted units. The main part of the paper is structured as follows. Section 2 addresses properties of the trust relation itself, making only informal reference to notions of information. Building on a simple trustworthiness relation (2.1), rational trust attitudes are identified and ordered according to strength (2.2, 2.3), and ordered in a tree structure of "fallbacks" (2.4). Section 3 employs this structure to provide an account of trust in terms of default conditionals. Notions of information, as provided by individual sources as well as collections of sources, are defined in 3.1. The prioritized default logic $\text{Æ}_\top$ is briefly presented in section 3.2. The defaults approach is then made explicit in section 3.3, which presents a method for expressing trust attitudes as formulae of $\text{Æ}_\top$.

For the presentation of the core theory, we assume that the information provided by sources is expressed in propositional logic. However, the theory is equally applicable if one wants to use a more, or less, complex language. Looking forward, section 3.4 outlines how the analysis can be applied to multi-agent doxastic logic, to enable the representation of doxastic agents with varying degrees of trust that the beliefs of other agents are true.

The expression of trusting attitudes in terms of prioritized defaults provides an answer to the following non-trivial question: Given that we possess a body of antecedent knowledge, and are provided with information from a set of variously trusted sources, what is it reasonable to believe?

This work builds on two main sources. For the theory of trustworthiness, the most important is the work of John Cantwell [1, 2], in which the basic relation of trustworthiness is defined in a way that is close to the one given here. For the aspects that relate to default inference and belief, the prioritized belief logic Æ [9, 10, 12], which is closely related to that of [7], has been the primary source of reference.

We consider the following to be guiding principles for what follows.

Given a collection of sources, what all sources agree on is at least as trustworthy as what only some agree on. (1)

If some unit x is trusted, and y is at least as trustworthy as x, then rationality demands that y should be trusted too. (2)

Accept information from a trusted unit as true, unless it is inconsistent with what you have already accepted. (3)

2 A trustworthiness relation

2.1 The basic pre-order on information sources

Let $\mathfrak{S}$ be a (possibly empty) finite set of *sources*. The *trustworthiness relation* $\trianglelefteq$ is a relation between subsets of $\mathfrak{S}$; we will often refer to these as *source units*. A source unit is an entity that is capable of providing information, as follows: A singleton unit $\{a\}$ provides exactly what the single source a does. A non-singleton unit provides only what follows from the contribution of every member. Informally, think of a non-singleton source unit as making a "common statement", i.e., the strongest that its members all agree on.

Notation: Small Latin letters a, b, c denote sources, small variable letters x, y, z range over source units, capital Latin letters A, B, C denote particular sets of source units, and capital variable letters X, Y, Z range over arbitrary sets of source units. We will sometimes have to collect sets of source units, for which we shall use capital Greek letters $\varGamma, \varDelta$.

We assume that the trustworthiness relation is reflexive and transitive (a *pre-order*). Two source units x and y may be *trustworthiness-equivalent*, written $x \sim y$.

$$x \sim y \quad =_{\text{def}} \quad x \trianglelefteq y \text{ and } y \trianglelefteq x \tag{4}$$

We write $x \lhd y$ to express that y is strictly more trustworthy than x.

$$x \lhd y \quad =_{\text{def}} \quad x \trianglelefteq y \text{ and not } x \sim y \tag{5}$$

Source units that are unrelated by $\trianglelefteq$ will be called *independent*, denoted $x \wr y$. Intuitively, we interpret independence as a consequence of lack of knowledge; neither of $x \rhd y$, $x \lhd y$, and $x \sim y$ is known to obtain. If no two source units are independent, we say $\trianglelefteq$ is *connected*.

We assume that every source, however it is combined with other sources, makes a non-negative contribution of information. Together with (1), this implies that enlargement of a source unit with new members may never yield a unit that provides a stronger set of information. Hence, a unit will be at least as trustworthy as every unit that it contains as a subset. This motivates taking the following principle, which we will occasionally refer to as *monotonicity*, to be valid.

$$x \trianglelefteq x \cup y\,. \tag{6}$$

It follows that for each source unit x, the following hold.

$$x \trianglelefteq \mathfrak{S}\,, \tag{7}$$

$$\emptyset \trianglelefteq x\,. \tag{8}$$

To see why (7) is valid, note that $\mathfrak{S}$ only provides information which is common to, is agreed upon, by all the sources. At the other extreme, we stipulate that the empty set is a limit case that always provides inconsistent information, motivating (8).

In referring to particular source units in examples we will consistently simplify notation by omitting brackets: $a \lhd bc$ is, e.g., shorthand for $\{a\} \lhd \{b, c\}$. Likewise, the set $\{\{a\}, \{a, b\}\}$ will be denoted a, ab. Observe that the symbol a should, depending on the context, either be taken as a reference to the source a or to the singleton source set $\{a\}$ or to the singleton source set collection $\{\{a\}\}$.

2.2 The poset of trust-equivalent source units

To have an *attitude* of trust, given some $\mathfrak{S}$, is to trust a (possibly empty) set of source units. In the following, we will allow ourselves to talk about attitudes as being the sets of source units themselves, and to say that a source unit is "included" in an attitude of trust, meaning that that source unit is among those trusted. The empty set represents the attitude of placing trust in none of the sources.

Given a trust relation $\lhd$, we can distinguish those trust attitudes that respect the relation. The relevant principle is expressed in rule (2), that x may only be trusted if every $y \unrhd x$ is trusted as well. We will in this section identify the *permissible* trust attitudes according to this principle.

We will use the following standard terminology. In a *poset* $(S, \leq)$ the $\leq$-relation is reflexive, transitive and anti-symmetric. The poset has a unique cover relation $\prec$, defined as $x \prec y$ iff $x < y$ and $x \leq z < y$ implies $z = x$. $C \subseteq S$ is an *antichain* if every two distinct elements in C are incomparable by $\leq$. Note in particular that $\emptyset$ is an antichain. Every subset of S has $\leq$-minimal elements, and the set of these elements is an antichain. $\uparrow C$ denotes an *up-set*, defined as $\{x \mid (\exists y \in C)(y \leq x)\}$. The set of antichains in a poset is isomorphic to the set of up-sets under set inclusion.

If an attitude of trust includes a source set x, but not an equivalently trustworthy source set y, then the attitude is not permissible. This motivates a focus on the equivalence classes of $\mathfrak{S}$ modulo $\sim$. Where $x \subseteq \mathfrak{S}$,

$$[x] \quad =_{\text{def}} \quad \{y : x \sim y\} \tag{9}$$

Let $\dot{\mathfrak{S}}$ be the set of all equivalence classes of $\mathfrak{S}$ modulo $\sim$. We will say a set of sources x is *vacuous* with regard to trustworthiness if $x \in [\emptyset]$. In the extreme case that every set of sources is a member of $[\emptyset]$, the trustworthiness relation itself is said to be vacuous.

Where X and Y are in $\dot{\mathfrak{S}}$, define a relation $\dot{\lhd}$ of relative strength between them as follows.

$$X \mathrel{\dot{\lhd}} Y \quad =_{\text{def}} \quad (\exists x \in X)(\exists y \in Y)(x \lhd y) \tag{10}$$

Let $X \mathrel{\dot{\unlhd}} Y$ designate $X \mathrel{\dot{\lhd}} Y$ *or* $X = Y$ and let $X \mathrel{\dot{\wr}} Y$ designate independence.

Lemma 1. *$(\dot{\mathfrak{S}}, \dot{\unlhd})$ is a poset in which $[\emptyset]$ is the unique minimum and $[\mathfrak{S}]$ the unique maximum. $(\dot{\mathfrak{S}}, \dot{\unlhd})$ is a linear order iff $(\wp\,\mathfrak{S}, \unlhd)$ is connected.*

Proof. Monotonicity entails the unique minimum and maximum. The other properties follow easily from the construction of $(\dot{\mathfrak{S}}, \dot{\unlhd})$.

Example 1. Assume that the set of sources $\mathfrak{S}$ contains just a and b, and that $a \lhd ab$, $b \lhd ab$, $\emptyset \lhd a$, and $\emptyset \lhd b$ (i.e., the source units a, b, and ab are non-vacuous, and ab is more trustworthy than both a and b). The following figure shows Hasse diagrams of the poset $(\dot{\mathfrak{S}}, \dot{\unlhd})$, given 1. $a \lhd b$, 2. $a \sim b$, and 3. $a \wr b$.

1.
$$\begin{tikzcd} ab \arrow[d, dash] \\ a \arrow[d, dash] \\ b \arrow[d, dash] \\ \emptyset \end{tikzcd}$$

2.
$$\begin{tikzcd} ab \arrow[d, dash] \\ a, b \arrow[d, dash] \\ \emptyset \end{tikzcd}$$

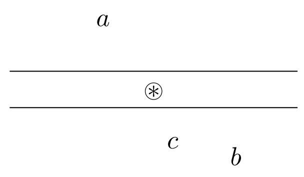
3.
$$\begin{tikzcd} & ab \arrow[dl, dash] \arrow[dr, dash] & \\ a \arrow[dr, dash] & & b \arrow[dl, dash] \\ & \emptyset & \end{tikzcd}$$

Relation 1. requires information provided by a to take precedence over information provided by b. Relation 2. emerges from taking a to precisely as reliable as b: it is only rational to accept a's contribution given that b's is accepted as well (in the event that a and b contradict each other, it is ruled out that either can be trusted separately). Relation 3. reflects a situation in which less is known about the relative trustworthiness of a and b than in 1. and 2, i.e., neither is known to be better or equivalent to the other. With this relation, trusting b but not a is not irrational; so the range of admissible attitudes is wider. In particular, where the information a provides is incompatible with what b provides, the relation doesn't rule out making a *choice* of trusting just one of the two.[1] Compared to 1. and 2., this relation offers more freedom, but less guidance.

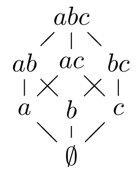
The following example, which is developed further in later sections, applies the theory to a reasonably realistic scenario.

Example 2 (Traffic accident). A traffic accident has occurred. We have been assigned the task of finding witnesses, assessing their relative trustworthiness, gathering their statements on what came to pass, weighing the evidence according to trustworthiness and finally presenting an account of the accident according to a reasonable standard (threshold) of trust.

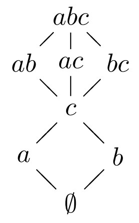
Assume, for this example, that the criterion according to which sources are deemed trustworthy or not is their viewpoint relative to the incident, and that we are provided with a drawing (right), illustrating the accident $\circledast$ and the positions of the witnesses. At the outset, we know that there are three witnesses, a, b, and c, but nothing about their respective trustworthiness. Making no prior assumptions, we start out with the weakest possible trust relation (0. below).



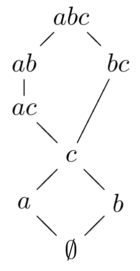
0.



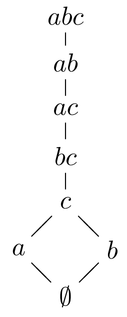
1.



2.



3.



By applying information provided by the drawing, we are able to considerably strengthen the trust relation. We will consider a sequence of three steps.

[1] When the case arises that a and b contradict each other, a choice will implicitly favour a revision of the trust relation to be like 1. or 2. If the subject opts to trust a over b, 1. is favored; if neither, this favors 2.

1. Seeing that c was closer to where the accident took place than the others, we take c to be more trustworthy than both: $a \lhd c$ and $b \lhd c$.

2. Because a and b are farther apart than a and c, their viewpoints are likely to be more divergent. Whatever can be observed from widely different perspectives is likely to hold true. Therefore, we will assume $ac \lhd ab$.

3. Because b and c are close together, we add $bc \lhd ac$ as well.

We choose to make no further additions to the relation. In particular, we refrain from making a judgment whether a is more trustworthy than b, or vice versa, or just as trustworthy as b: we consider a and b to be independent. This means it will be consistent with the trust relation to make a choice between which of a and b to trust. If they should happen to contradict each other, our lack of knowledge as to which is more trustworthy then presents us with the option to trust just one of the two.

Note that c is more trustworthy than b in 2. and 3., but that the relationship is not preserved when combined with a ($ac \lhd ab$ holds). Indeed, the following substitution principles are not valid; given $z[y/x] = (z \setminus x) \cup y$,

$$\text{If } x \lhd y \text{ and } x \subset z, \text{ then } z \lhd z[y/x]$$
$$\text{If } x \sim y \text{ and } x \subset z, \text{ then } z \sim z[y/x]\,.$$

2.3 A lattice of trust levels

We know from Lemma 1 that $(\dot{\mathfrak{S}}, \dot{\unlhd})$ is a poset. Given the poset it is straightforward to identify the permissible trust attitudes: a trust attitude is permissible if it is an up-set in $(\dot{\mathfrak{S}}, \dot{\unlhd})$. Technically, we will represent an attitude by its set of minima, or equivalently, by an antichain in the partial order $(\dot{\mathfrak{S}}, \dot{\unlhd})$. We define the set $\mathfrak{T}$ of permissible trust attitudes as follows,

$$\mathfrak{T} = \{\cup\mathit{\Gamma} \mid \mathit{\Gamma} \text{ is an antichain in } (\dot{\mathfrak{S}}, \dot{\unlhd})\}$$

We will use the symbol $\curlywedge$ to denote the attitude that no source unit is trusted, $\cup\emptyset$.

There is a natural relation of strength between permissible trust attitudes. Having a weak trust attitude means trusting only what many sources agree on, or perhaps none; a strong attitude means trusting many sources, or perhaps all. Let $\mathit{\Gamma}$ and $\mathit{\Delta}$ be antichains in $(\dot{\mathfrak{S}}, \dot{\unlhd})$. Then we define

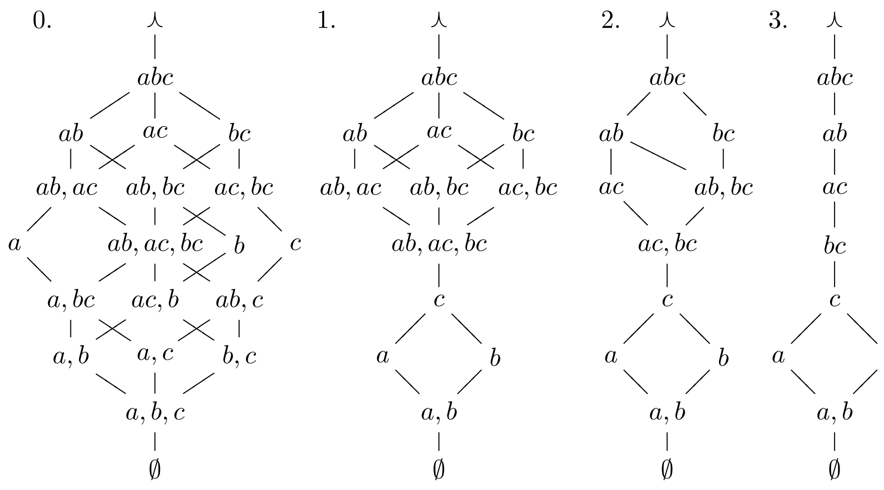
$$\cup\mathit{\Gamma} \leq \cup\mathit{\Delta} \text{ iff } \uparrow\!\mathit{\Delta} \subseteq \uparrow\!\mathit{\Gamma}.$$

By definition, $\curlywedge$ is $\leq$-maximal in $\mathfrak{T}$. This is natural, as the corresponding attitude of trusting no source unit will always have a maximal degree of reliability. Ordered by $\leq$, the members of $\mathfrak{T}$ form a lattice in which lesser nodes represent stronger trust attitudes. It is natural to talk about the permissible trust attitudes as corresponding to a hierarchy of degrees of trust. We shall hence occasionally refer to $\mathfrak{T}$ as the set of *trust levels*.

In the lattice $(\mathfrak{T}, \leq)$ $A < B$ intuitively means that B is a level of trustworthiness that is genuinely greater than A. Let $\sqcap$ denote meet and $\sqcup$ denote join.

Then $A \sqcup B$ is the weakest trust level that is at least as strong as both A and B; if A and B are not comparable by $\leq$, then it is stronger. $A \sqcap B$ is the strongest trust level that is at least as weak as both A and B.

Example 3 (Lattices for example 2).



The lattice of trust levels makes explicit what the permissible trust attitudes are and how they are related with regard to strength. This can form the basis for choosing, in a given scenario, a *threshold* of trust: a level that is deemed sufficiently trustworthy. Setting a threshold may also be described in terms of *risk*. If $A < B$, then to choose A as the threshold of trust is to take a greater risk with regard to trusting sources than if B is chosen. Determining a threshold of trustworthiness amounts to fixing a “limit” of risk, to draw a line between what is trusted, and not trusted, in the non-relative sense of the word. For example, with a threshold at $A \sqcup B$, if A and B are comparable, risk is limited to what follows from trusting the more trustworthy of the two; if incomparable, then to the greatest degree of risk that represents comparably less risk than both A and B. To say that $A \sqcap B$ lies within the risk limit means that A and B are both considered reliable (i.e., that all source units in A and B provide only true information).

A threshold of trust can be conveniently specified by reference to the source units trusted. Observe that each member of $\dot{\mathfrak{S}}$ is a member of $\mathfrak{T}$. Therefore, any expression using members of $\dot{\mathfrak{S}}$ (i.e., equivalence classes of source units), $\sqcap$ and $\sqcup$ denotes a unique level of trust.

Example 4 (Threshold for example 3). Say that we adopt the attitude to “trust all that *ab* and *ac* deliver, as long as it is confirmed by *bc*” as a threshold. This attitude is expressible as $([ab] \sqcap [ac]) \sqcup [bc]$. Given relations 0., 1., and 2., the attitude amounts to trusting only what a, b, and c agree on, because $(ab \sqcap ac) \sqcup bc = (ab, ac) \sqcup bc = abc$. With the stronger relation 3., it denotes the level ab.

2.4 A tree of *fallbacks* for broken trust

The core of a default conception of relative trust in information sources is the default rule (3) to accept what you are told, unless it is in conflict with what you already know We presently interpret this rule with respect to relative trust. Let us consider a trusting subject that has only permissible trust attitudes. In the non-relative sense of "trust", $\Yup$ is always trusted, and a level X is trusted, on condition that every $Y \geq X$ is also trusted, by default.

Now, if trusting at a level X is inconsistent with trusting at a superior level Y, trust at X is broken; X is not trustable. This will obtain whenever information provided at X is inconsistent with antecedent knowledge, or with information accepted at a superior level. The significance of of trusting at X should then be identified with trusting some superior, trustable level; call this the *fallback* of X. The fallback, as the value of a blocked default, is the key notion that allows us to view relative trust as a default attitude.

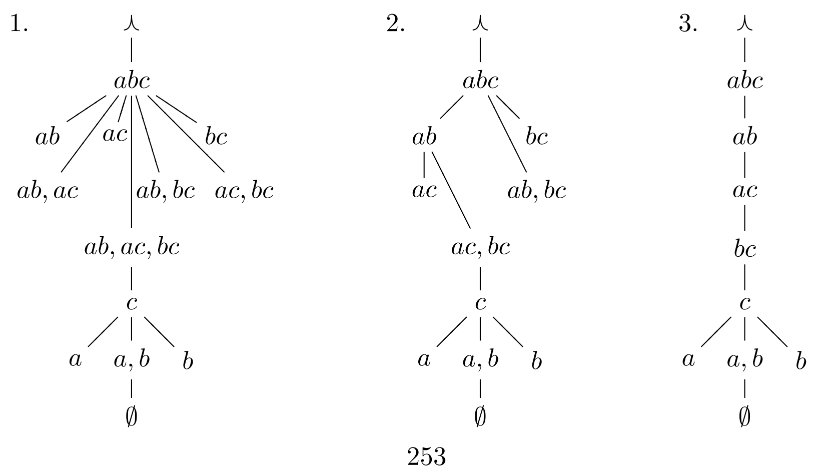
Let X be an element of $\mathfrak{T}$ different from $\Yup$, and let Γ be the $\leq$-cover of X. Given that Γ is singleton, we straightforwardly identify $\bigcup \Gamma$ as the appropriate fallback of X. Where not, note that by construction of the lattice, X is a level composed of a set of simpler levels, the members of Γ. That trust is broken at X means some of these levels are not trustable. In this case, the fallback of X should be identified as a level with greater trustworthiness than every Y immediately superior to X. Let the fallback $\mathfrak{f}(X)$ of X be defined as

$$\mathfrak{f}(X) = \mathrm{lub}(\Gamma) \text{ in } (\mathfrak{T}, \leq)\,.$$

The fallback function is undefined for $\Yup$; otherwise every node has a unique fallback. $\Yup$, representing the trust level of antecedent knowledge, is always the fallback of $[\mathfrak{S}]$. Note that every path from the lattice maximum $\Yup$ to a trust level X must go through $\mathfrak{f}(X)$, and that $\mathfrak{f}(X)$ is the $\leq$-minimal node with this property.

The *fallback tree* $(\mathfrak{T}, \prec)$ is defined as the weakest relation such that for all $X \in \mathfrak{T}$, $\mathfrak{f}(X) \prec X$. It is easy to show that the fallback tree is indeed a tree with root $\Yup$.

Example 5 (Fallback trees for example 3).



3 Trust in terms of defaults

The aim of this section is to implement the default approach to the information trust model based on a function $\mathbb{I}$ which assigns propositional content to each source in $\mathfrak{S}$. The default interpretation of fallback trees is then encoded into the logic $Æ_\top$. Encoding fallback trees in $Æ_\top$ will allow us to give precise answers to questions such as, "which trustworthiness levels support a belief in a proposition ϕ?", and "is ϕ entailed by the beliefs of a given degree of trustworthiness?". $Æ_\top$ is a natural choice as representation language for default inferences. It allows a simple representation of ordered supernormal defaults theories as well as a natural extension to multi-agent languages.

The basic assignment of information to sources is a mapping from members of $\mathfrak{S}$ to expressions in a formal language. In section 3.1 we use the simple language of propositional logic to this end. However, there is no intrinsic reason for using this language to represent information, and one can easily conceive of using more complex languages for this purpose. Section 3.4 explores possibilities for using multi-agent langauges.

3.1 Information provided by sources

Basically the information interpretation of the trust model assigns formal expressions to each source in $\mathfrak{S}$. The assignment function $\mathbb{I}$ must then be extended to source units (sets of sources) and trust attitudes (sets of source units). To implement this we identify the corresponding operations of *agreement* and *consolidation* of information content. In propositional logic these operations will be implemented simply by means of disjunctions and conjunctions.

Let us denote the informational content of a source a in $\mathfrak{S}$ by $\mathbb{I}_a$, which is a formula of propositional logic. Intuitively, the information $\mathbb{I}_x$ provided by a source unit is defined to be the strongest proposition that every member of the unit supports – the strongest that the members all agree on. If $x = \{a_1, \ldots, a_n\},\ a_i \in \mathfrak{S}$, then $\mathbb{I}_x = \mathbb{I}_{a_1} \vee \cdots \vee \mathbb{I}_{a_n}$. The value of $\mathbb{I}_\emptyset$, on the common understanding of 0-ary disjunctions, will be assumed to be the propositional falsity constant $\bot$. The empty set hence gives a contribution which is always unacceptable.

Define the *consolidated* informational contribution of $x_1, \ldots, x_n \subseteq \mathfrak{S}$ as $\mathbb{I}_X = \mathbb{I}_{x_1} \wedge \cdots \wedge \mathbb{I}_{x_n}$. That is, we define the informational contribution of a set of source units as the strongest consequence that would follow from taking each unit as a source of evidence. Observe in particular that $\mathbb{I}_{[\emptyset]}$ will always be $\bot$. By convention $\mathbb{I}_\curlywedge$ is $\top$.

3.2 Intermezzo: The Logic $Æ_\top$

$Æ_\top$ is an "Only knowing" logic, generalizing the pioneering system of Levesque [7] with language constructs for the representation of various degrees of confidence for a doxastic subject.

The object language of $Æ_\top$ extends the language of propositional logic by the addition of modal operators: $\Box$ (necessity) and modalities B_k (belief) and

C_k (co-belief) for each k in a finite index set I. The index set represents the distinct degrees of confidence and comes along with a partial order which gives the indices relative strength. $\mathsf{b}_k\,\varphi$ is defined as $\neg\,\mathsf{B}_k\,\neg\varphi$ and denotes that φ is compatible with belief at degree of confidence k.

A formula φ is *completely modalized* if every occurrence of a propositional letter occurs within the scope of a modal operator and *purely Boolean* if it contains no occurrences of modal operators. The "all I know at k" expression $\mathsf{O}_k\,\varphi$ abbreviates $\mathsf{B}_k\,\varphi \wedge \mathsf{C}_k\,\neg\varphi$, meaning that *precisely* φ is believed with degree of confidence k. A formula of the form $\bigwedge_{k\in I} \mathsf{O}_k\,\varphi_k$ is called an O_I*-block*. If each φ_k is purely Boolean, the O_I-block is said to be *prime*.

$\text{Æ}_\top$ is a special instance of the system Æ_ρ introduced in [8] and further analyzed and motivated in [12]; the references contain in particular an axiomatization, a formal semantics and proofs of soundness, completeness and the finite model property. A particularly strong property of $\text{Æ}_\top$ is the Modal Reduction Theorem: for each O_I-block φ^I and for some $m \geq 0$, there are prime O_I-blocks $\psi_1^I, \ldots, \psi_m^I$ such that $\vdash \varphi^I \equiv (\psi_1^I \vee \cdots \vee \psi_m^I)$.[2]

A prime O_I-block determines the belief state of the agent in a unique and transparent way; if such a formula is satisfiable, it has essentially only one model. A non-prime O_I-block only implicitly defines the belief state and has in general a number of different models. The Modal Reduction Theorem relates an implicit belief representation to an explicit representation by a provable equivalence. To determine whether $m > 0$ in the statement of the theorem is Σ_2^p-hard.

If there is only one degree of confidence, $\text{Æ}_\top$ is equivalent to Levesque's system of only knowing, for which there is a direct correspondence between a stable expansion in autoepistemic logic and a prime formula $\mathsf{O}\,\varphi$. A prime O_I-block is a natural generalization of the notion of stable expansion to a hierarchical collection of expansions.

3.3 Encoding the fallback tree as defaults in $\text{Æ}_\top$

We now describe how to use a fallback tree to extract information, both between contributions of the sources, which may be more or less mutually compatible, and between these contributions and a set of antecedently given information.

To facilitate the discussion let us say that a fallback tree is *information labelled* if each node X in the tree is labelled with $\mathbb{I}_X$. The labels express the information contribution attached to the trust level X.

We will assume that a *knowledge base*, denoted κ, is given with unconditional trustworthiness. Informally, say that (precisely) κ, a formula of propositional logic, is believed with full conviction. The notion of trustworthiness is directly relevant to the notions of confidence and belief, as is clear by the simple observation that information stemming from highly trustworthy sets of sources will be considered reliable with a greater degree of confidence than that which is

[2] In the sequel $\vdash$ denotes the provability relation of $\text{Æ}_\top$ (which extends the provability relation of classical logic).

provided by less trustworthy sources. Following the default interpretation formulated in principle (3), we can define a simple procedure which reveals what information may reliably be said to be supported at each level of trustworthiness. Define the following formula by induction over the fallback tree.

$$\begin{aligned}\beta_{\curlywedge} &= \kappa \\ \beta_X &= \begin{cases}\beta_{\mathfrak{f}(X)} \wedge \mathbb{I}_X & \text{if } \beta_{\mathfrak{f}(X)} \wedge \mathbb{I}_X \text{ is PL-consistent,} \\ \beta_{\mathfrak{f}(X)} & \text{otherwise.}\end{cases}\end{aligned}$$

Then β_X denotes what a rational agent should believe at a degree of confidence corresponding to the trust attitude X.

The modal logic $Æ_{\top}$ is suitable for the representation of fallback trees and the associated default principle. In the encoding we use the set of trust levels $\mathfrak{T}$ as the index set which individuates modalities in the language of $Æ_{\top}$. Let $(\mathfrak{T}, \prec)$ be the fallback tree and $\prec^*$ be the reflexive, transitive closure of $\prec$. For $X \in \mathfrak{T}$ we define

$$\delta_X = \mathsf{b}_X\, \mathbb{I}_X \supset \mathbb{I}_X\ .$$

Note that δ_X is equivalent to $\neg\, \mathbb{I}_X \supset \mathsf{B}_X \neg\, \mathbb{I}_X$, i.e., should φ be false, the subject will believe that it is. We will refer formulae of this form as *default conditionals* when they occur within a modal O-context, since the conditional then has the force of formalizing the property corresponding to the statement "the proposition $\mathbb{I}_X$ holds by default".

The default interpretation of the default structure is formalized by the following encoding:

$$\begin{aligned}[\![\mathfrak{T}, \prec, \kappa]\!]_{\curlywedge} &= \mathsf{O}_{\curlywedge}\, \kappa \\ [\![\mathfrak{T}, \prec, \kappa]\!]_X &= \mathsf{O}_X(\kappa \wedge \textstyle\bigwedge_{Y \prec^* X} \delta_Y) \\ [\![\mathfrak{T}, \prec, \kappa]\!] &= \textstyle\bigwedge_{X \in \mathfrak{T}} [\![\mathfrak{T}, \prec, \kappa]\!]_X\end{aligned}$$

The encoding is structurally similar to the encoding of ordered default theories into $Æ_{\top}$ in [4].

Theorem 1. $\vdash [\![\mathfrak{T}, \prec, \kappa]\!] \equiv \bigwedge_{X \in \mathfrak{T}} \mathsf{O}_X\, \beta_X$.

Proof. The proof uses simple properties from the model theory of $Æ_{\top}$, cf. [12]. In an $Æ_{\top}$ model M all points agree on the truth value of every completely modalized formula. We will hence use the notation $M \models \varphi$ whenever a completely modalized φ is satisfied at some point in M. We use the following two facts in the proof. Let M satisfy $\mathsf{O}_X\, \varphi$ for an index X.

1. If M satisfies $\mathsf{O}_X\, \psi$, then $\varphi \equiv \psi$ is true at every point in M.
2. If φ and ψ are purely Boolean, M satisfies $\mathsf{b}_X\, \psi$ iff $\varphi \nvdash \neg\psi$.

We show, by induction on X, the more general result that for any $Z \in \mathfrak{T}$

$$\vdash \textstyle\bigwedge_{X \prec^* Z} [\![\mathfrak{T}, \prec, \kappa]\!]_X \equiv \textstyle\bigwedge_{X \prec^* Z} \mathsf{O}_X\, \beta_X\ .$$

The base case is trivial. For the induction step, it is sufficient to show that $M \models [\![\mathfrak{T}, \prec, \kappa]\!]_X \equiv \mathsf{O}_X\, \beta_X$ for any $\text{Æ}_\top$-model satisfying both $[\![\mathfrak{T}, \prec, \kappa]\!]_{\mathfrak{f}(X)}$ and $\mathsf{O}_{\mathfrak{f}(X)}\, \beta_{\mathfrak{f}(X)}$. By 1, every such model M satisfies

$$M \models (\kappa \wedge \bigwedge\nolimits_{Y \prec^* \mathfrak{f}(X)} \delta_Y) \equiv \beta_{\mathfrak{f}(X)} \ \ .$$

Thus $M \models [\![\mathfrak{T}, \prec, \kappa]\!]_X \equiv \mathsf{O}_X(\beta_{\mathfrak{f}(X)} \wedge \delta_X)$. It only remains to show

$$M \models \mathsf{O}_X(\beta_{\mathfrak{f}(X)} \wedge (\mathsf{b}_X\, \mathbb{I}_X \supset \mathbb{I}_X)) \equiv \mathsf{O}_X\, \beta_X \ \ .$$

But since $M \models \mathsf{O}_{\mathfrak{f}(X)}\, \beta_{\mathfrak{f}(X)}$, it follows directly from the definition of β_X and 2 that $M \models \mathsf{b}_X\, \mathbb{I}_X$ iff $\beta_{\mathfrak{f}(X)} \not\vdash \neg\, \mathbb{I}_X$, and we are done. $\square$

The theorem shows that the encoding of a node X and its information content can be reduced to the $\mathsf{O}_{\mathfrak{T}}$-block $\bigwedge_{X \in \mathfrak{T}} \mathsf{O}_X\, \beta_X$ within the logic itself, where at each node X in the tree the formula β_X is the proposition that the rational agent will entertain at this level of trust.

Example 6 (Example 5, with information). The witnesses a, b, and c are interviewed for their accounts of the accident scenario. We assign content to propositional variables as follows: p = The green car was veering; q = There was a cat in the road; r = The red car was veering; s = The red car was speeding. The following figure records the witnesses' statements (left), and the resulting post-evaluation propositions at each trust level decorate the fallback tree (3.).[3]

$$\begin{array}{lcl} a & : & q \wedge (r \vee s) \\ b & : & p \wedge \neg q \\ c & : & p \wedge r \end{array}$$

3.

$$\begin{tikzcd}
 & \curlywedge \ : p \vee r \arrow[d, dash] & \\
 & abc \ : (p \vee r) \wedge (q \wedge (r \vee s) \vee p \wedge (\neg q \vee r)) \arrow[d, dash] & \\
 & ab \ : (p \vee r) \wedge (q \wedge (r \vee s) \vee p \wedge \neg q) \arrow[d, dash] & \\
 & ac \ : (p \vee r) \wedge (q \wedge s \vee (q \vee p) \wedge r) \arrow[d, dash] & \\
 & bc \ : p \wedge r \arrow[d, dash] & \\
 & c \ : p \wedge r \arrow[dl, dash] \arrow[dr, dash] \arrow[dd, dash] & \\
p \wedge (r \wedge q) : \ a & & b \ : p \wedge r \wedge \neg q \\
 & a, b \ : \bot \arrow[d, dash] & \\
 & \emptyset \ : \bot &
\end{tikzcd}$$

Noteworthy features:

- a and b may not both be fully trusted, but choosing either is consistent.
- The proposition s, which figures as a disjunct in a's account, is eliminated from the node bc onwards.

[3] Formulae computed using *The Logics Workbench*, `http://www.lwb.unibe.ch/`.

- For nodes a, b and $\emptyset$, the value $\perp$ is displayed to emphasize their inconsistency. These nodes will actually take values from the consistent fallback node c, i.e., $p \wedge r$.

3.4 From information sources to doxastic agents

There is no intrinsic reason to use the language of propositional logic to represent the information delivered by sources. This section addresses the use of multi-modal languages for this purpose. The expressive power of such languages is needed in cases where the sources deliver information about agents; typically, about what the agents believe. To generalize the approach of section 3.3 we need to extend the language of $\text{Æ}_\top$ such that it extends the information representation language.

The logic $\text{Æ}_\top$ has been extended to a multi-modal language. An interesting proof-theoretical property of this extension of $\text{Æ}_\top$ is that it has a sequent calculus formulation which admits constructive cut-elimination and hence cut-free proofs; this is proved in [11] for a multi-agent language in which the beliefs of each subject are represented relative to different degrees of confidence. A Kripke semantics for the logic has been presented in [13].[4]

Let us assume that the modalites in the multi-agent language is defined by a collection $I_0, \ldots, I_m$ of index sets, one for each agent. The indices in each index set are partially ordered, while two indices in different index sets are unrelated.

The notion of an O_I-block transfers to the multimodal langage: An O_{I_j}-block is a formula $\bigwedge_{k \in I_j} \mathsf{O}_k\, \varphi_k$. If each formula φ_k is I_j-objective, i.e. all occurrences of a I_j-modality occurs within the scope of a modality which belongs to another agent, the O_{I_j}-block is prime. An O_I-block can now be defined as a conjunction of O_{I_j}-blocks, one for each agent. Given these concepts the Modal Reduction Theorem transfers to the multi-modal logic.

Let us first assume that the sources deliver information about the beliefs of agents $\alpha_1, \ldots, \alpha_m$ without being agents themselves, i.e. they do not deliver information about other sources, or about themselves, or about the observer who collects the information. Assume also that the beliefs of these agents are represented in the multi-modal system K45$_m$, i.e. a sublanguage of multi-modal $\text{Æ}_\top$, so that the $\mathbb{I}$ function now delivers K45$_m$ formulae.

The index sets for the multi-modal $\text{Æ}_\top$-representation are $\mathfrak{T}, \{\alpha_1\}, \ldots, \{\alpha_m\}$. The assumption that no α_i are sources implies that we can use the same simple functions for agreement and consolidation as introduced for propositional logic in section 3.1. It is now straightforward to establish Theorem 1 for the language at hand.

Example 7 (Modal information). A simple case in which sources provide formulae in a modal language. Let the trustworthiness relation be given as in example 1, relation 3. Let the knowledge base be empty, and assign information to sources

[4] The semantics has been given for a multi-agent language without confidence levels. An extension to the languge addressed in this section is straightforward.

as below (left). The fallback tree shows the outcome of evaluation (right). Here, trusting what a and b agree on (source unit ab) implies accepting that agent 1 has a full belief regarding p. Trusting both sources (node a, b) implies accepting that 1 is inconsistent.

$$\begin{array}{lcl} a & : & \mathsf{B}_1\, p \\ b & : & \mathsf{B}_1\, \neg p \end{array}$$

$$\begin{tikzcd}
 & \curlywedge \arrow[d, dash] & \\
 & ab \ : \mathsf{B}_1\, p \lor \mathsf{B}_1\, \neg p \arrow[dl, dash] \arrow[dr, dash] \arrow[dd, dash] & \\
\mathsf{B}_1\, p : \ a & & b \ : \mathsf{B}_1\, \neg p \\
 & a, b \ : \mathsf{B}_1(p \land \neg p) \arrow[d, dash] & \\
 & \emptyset \ : \bot &
\end{tikzcd}$$

If the information sources are themselves agents, the situation is at once much more complex, and we propose this to other researchers in the community as an interesting and challenging application of multi-modal logics. One problem is that we can no longer implement agreement and consolidation by means of simple Boolean operations. In some cases we may use the notion of "group belief" for agreement and "distributed belief" for consolidation (see e.g. [5]).

However, we can also use the full expressive power of multi-modal $\text{Æ}_{\top}$ to specify very complex formulae delivered by each agent, in which case these operators are no longer sufficient for this purpose. We plan to address this in a follow-up paper.

4 Related work

The present account of trustworthiness generalizes and clarifies the approach introduced by John Cantwell [1]. Our approach improves on Cantwell's by making a clear separation between the notion of trustworthiness on the one hand, and information and belief on the other, which allows for the notion of trustworthiness level to be separated from a given model. Furthermore, the present theory gives informative results for various weak kinds of trustworthiness relations that yield vacuous output on Cantwell's approach.[5]

In this paper, no attempt has been made to give a general account of the basic non-relative notion of trust; for this, see Jones [6]. We intend to apply the present theory of relative trustworthiness to Jones' analysis of trust in a forthcoming paper. We also wish to explore the complex subjects of construction and revision of trustworthiness relations in the future.

[5] Cantwell incorporates his theory of trustworthiness into a theory of *belief revision*. This is an application that we have not gone into.

Bibliography

[1] John Cantwell. Resolving conflicting information. *Journal of Logic, Language, and Information*, 7:191–220, 1998.

[2] John Cantwell. *Non-Linear Belief Revision.* Doctoral dissertation, Uppsala University, Uppsala, 2000.

[3] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, 2 edition, 2002.

[4] Iselin Engan, Tore Langholm, Espen H. Lian, and Arild Waaler. Default reasoning with preference within only knowing logic. Submitted for publication.

[5] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about Knowledge.* MIT Press, Cambridge, Mass., 1995.

[6] Andrew J. I. Jones. On the concept of trust. *Decision Support Systems*, 33:225–232, 2002.

[7] Hector J. Levesque. All I know: A study in autoepistemic logic. *Artificial Intelligence*, 42:263–309, 1990.

[8] E. H. Lian, T. Langholm, and A Waaler. Only knowing with confidence levels: Reductions and complexity. In *JELIA 2004*, Logics in Artificial Intelligence, 9th European Conference, pages 500–512, Lisbon, Portugal, 2004.

[9] K. Segerberg. Some modal reduction theorems in autoepistemic logic. *Uppsala Prints and Preprints in Philosophy*, 1995.

[10] Arild Waaler. *Logical Studies in Complementary Weak S5.* Doctoral thesis, University of Oslo, Oslo, 1994.

[11] Arild Waaler. Consistency proofs for systems of multi-agent only knowing. *Advances in Modal Logic*, 2005.

[12] Arild Waaler, Johan W. Klüwer, Tore Langholm, and Espen H. Lian. Only knowing with degrees of confidence. Submitted for publication, 2005.

[13] Arild Waaler and Bjørnar Solhaug. Semantics for multi-agent only knowing (extended abstract). Submitted for publication, 2005.

The First Contest on Multi-Agent Systems based on Computational Logic

Mehdi Dastani[1] and Jürgen Dix[2]

[1]Utrecht University
P.O.Box 80.089, 3508 TB Utrecht, The Netherlands
`mehdi@cs.uu.nl`
[2]Clausthal University of Technology
Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld, Germany
`dix@tu-clausthal.de`

Abstract. This is a short report about the first contest of Multi-Agent Systems (MASs) that are based on computational logic. The CLIMA workshop series (which started in 1999) is a forum to discuss techniques, based on computational logic, for representing, programming, and reasoning about Multi-Agent Systems in a formal way. Now in its it seventh year, it was felt that organising a competition for evaluating MASs based on computational logic is appropriate. The authors took on this task, which turned out to be quite difficult under the given time frame. We believe that this first competition is a (modest) first step towards (1) collecting important benchmarks, (2) identifying advantages/shortcomings and, finally, (3) analysing promising techniques of using Computational Logic in Multi-Agent Systems.

1 Introduction

Multi-Agent Systems are beginning to play an important role in todays software development: The *International Journal of Agent-Oriented Software Engineering (IDOSE)*[1], the *International Workshop on Agent-Oriented Software Engineering (AROSE)*[2] and the *International Joint Conference on Autonomous Agents and Multi Agent Systems* are just examples for that trend.

The development of MASs requires efficient and effective solutions for different problems which can be classified into two classes: the problems related to (1) the development of individual agents and (2) the development of their interactions. Typical problems related to individual agents are how to specify, design and implement issues such as *autonomy*, *pro-active/reative behaviour*, *perception and update of information*, *reasoning and deliberation*, and *planning*. Typical problems related to the interaction of individual agents are how to *specify*, *design and implement* issues such as *communication*, *coordination*, *cooperation* and *negotiation*.

This competition is a first attempt to stimulate research in the area of MASs by

1. identifying key problems, and
2. collecting suitable benchmarks

that can serve as milestones for testing new approaches and techniques from computational logic. While there already exist several competitions in various parts of artificial intelligence (theorem proving, planning, robo-cup, etc) and, lately, also in specialised areas in agent systems (trading agent competition (TAC) [3] and agentcities competitions [4]), the emphasis of this contest is on the use of *computational logic* in MASs. We believe that approaches and techniques of computational logic are essential for the development of MASs for at least two reasons: (1) logical approaches have proven to be very useful for specifying and modelling MASs in a precise manner, and (2) the specification and models can be executed.

We expect to promote the development of MASs by first identifying difficult problems and then finding solutions by comparing different approaches from computational logic for solving them. While this idea seems very appealing, it is not an easy task to come up with a particular scenario that serves as a basis for a contest. Such a scenario should be *generic* enough to be applicable for a wide range of techniques of computational logic, but it should also be *precise* enough so that different approaches can be tested and compared against each other.

2 Scenario description

This competition is organised as part of CLIMA and consists of developing MASs to solve a *cooperative task in a dynamically changing environment.* The environment of the MAS is a grid-like world where agents can move from one slot to a neighbouring slot if there is no agent already in that slot. In this environment, food can appear in all but one of these slots. The special slot, in which no food can appear, is considered as a depot where the agents can bring and collect their food. An agent can observe if there is food in the slot it is currently visiting. Initially, food can be placed in some randomly selected slots. During the execution, additional food can appear dynamically in randomly selected slots except the depot slot. The agents may have/play different roles (such as explorer or collector), communicate and cooperate in order to find and collect food in an efficient and effective way.

We have encouraged submissions that specify and design a MASs in terms of high-level concepts such as goals, beliefs, plans, roles, communication, coordination, negotiation, and dialogue in order to generate an efficient and effective solution for the above mentioned application. Moreover, the MAS implementations should be based on computational logic techniques (e.g., logic programming, formal calculi, etc.) and they should reflect their design in a direct and intuitive way.

We are completely aware of the fact that this scenario can also be attacked by completely different methods and approaches (e.g., based on machine learning, neural nets, etc.). In fact, we believe almost all scenarios can be modelled in various languages and programming paradigms. One important aim of this contest is to find out where exactly computational logic helps in solving particular problems and where other approaches are superior.

The challenge of this competition is thus to use computational logic techniques to provide implemented models for the abstract concepts that are used in the specification and design of MASs. These implemented models should be integrated to implement the above-mentioned application intuitively, directly, and effectively.

3 Submission format

A submission consisted of two parts. The first part is a description of analysis, design and implementation of a MAS for the above application. Existing MASs methodologies such as Gaia[12], Prometheus[11] and Tropos[9] can be used (not demanded) to describe the analysis and design of the system. For the description of the implementation, it should be explained how the design is implemented. This can be done by explaining, for example, which computational logic techniques are used to implement certain aspects of the MAS (including issues related to individual agents).

The second part is an (executable) implementation of the application. We did not demand any particular way (data format, algorithm, mechanism) to implement the system as long as it is implemented as a MAS and as long as the environment is a 20x20 grid. Moreover, it should be possible to configure the initial state of the environment to place food in arbitrary slots. During the execution food should appear automatically every 20 seconds in a randomly selected slot. The MAS will be run with 4 agents that are positioned initially at the corners of the grid. The implementation should be executable on a windows or linux machine.

3.1 Received Submissions

We have received four submissions for this first edition of the CLIMA contest. From the received submissions, only one submission did use an existing multi-agent methodology to develop a running system. Moreover, some submissions explain explicitly which techniques from computational logic are used to develop certain aspect of the MAS efficiently and effectively, while the use of computational logic techniques in other submissions is limited to the use of the Prolog language for the system implementation.

The submission from Carlos Cares analyses the scenario and designs a MAS in a systematic manner using Tropos, a well-known MAS methodology. The scenario is analysed in terms of multi-agent concepts and features such as actors, roles, beliefs, goals, plans, capabilities, commitments and resources. Based on these concepts, a system is designed in terms of instantiations of these concepts resulting a set of agents. In this submission, the Tropos methodology is extended with a Prolog implementation phase that allows the implementation of the Tropos-based architecture in terms of Prolog data structures such as lists, predicates, and rules.

The submission from Simon Coffey and Dorian Gaertner does not use an existing MAS methodology to develop their system. They provide directly a system architecture consisting of BDI agents that sense the grid environment to update their beliefs, evaluate their intentions, communicate with other agents, and select and execute actions. The agents are able to negotiate over their intentions to improve the efficiency of food collection. They also introduce different roles that agents can play such as the scouting role: a role for finding food. Based on different agents roles, they discuss a second system that consists of two types of agents: the agents that can only play the scouting role and the agents that can find and collect foods. Although in the proposed system the agents are static and can play only one role, they discuss the possibility of agents that can play different roles and can change their roles dynamically. In this submission, the designed system is implemented using Qu-Prolog that allows multi-threaded execution of agents.

The solution of Robert Logie, Jon G. Hall and Kevin G. Waugh consists of a purely reactive system of agents with no internal representation of the current state. Their system resembles Brookes subsumption architecture and has the notion of a *role* (or policy) at its core. Agents use certain roles and can switch between them when the environment changes. They use the idea of pheromone trails in order to find interesting and successful paths (their agents do not have a memory). Although their system does not seem to use computational logic in an extensive way, it has been motivated from research on normative reasoning in deontic logic. An interesting idea is that for more complex systems, this might lead to agents that develop and create new roles (in addition to those originally specified).

The final submission, by Eder Mateus Nunes Goncalves and Guilherme Bittencourt, concentrates on the notion of coordination between agents in a MAS. Each agent maintains a knowledge base and updates it accordingly. The underlying notion is a high-level petri net. Agents start cooperating with the agent closest to the food (once it has been found). The cooperation ends when the food is delivered at the depot. Messages are FIPA compliant. One of the main results is the influence of the appearance of new food (and the time it takes to store food in the depot) to the impact of cooperation between the agents. If the time interval for new food to appear is small with respect to the time it takes to store it, than cooperation pays off.

4 Winning Criteria

The criteria used to evaluate submissions and to select the winners are as follows:

1. Original, innovative, and effective application of computational logic techniques in solving specific multi-agent issues identified in this application.
2. The performance of the executable implementation. The performance of the executable implementation will be measured based on the amount of food

that is collected by the MAS in a certain period of time. All programs will be run on the same machine (Windows/Linux double boot machine).

3. The quality of the description of analysis, design and implementation of the MAS the elegance of its design and implementation, and the ease of installation and execution of the program.

The winner of this competition will be decided shortly before the CLIMA workshop and announced during the contest event of the workshop.

5 Conclusion

Given the very tight schedule (from the announcement to the submission deadline) we are more than satisfied with the four submissions. While these proceedings are in print, we are installing the systems and run them for several scenarios. The winner(s) will be announced at CLIMA VI in London.

We believe this contest will promote the use of techniques and approaches from computational logic to the development and implementation of MASs. Although the contributions for this contest may propose computational logic techniques and approaches that are specific for this particular scenario (application), they may be generalised and adopted to other MASmethodologies and programming languages. In particular, we believe that this contest will stimulate the use of computational logic techniques and approaches for research and design of programming languages that support the implementation of MASs in an effective and efficient manner.

There are several existing activities that aim at stimulating research and design of programming languages for MASs. Example of such activities are the *International Workshop on Programming Multi-Agent System* [5], the *AgentLink Technical Forum Groups on Programming Multi-Agent Systems* [6], and the various seminars and books dedicated to Multi-Agent Programming [7, 10, 8]. Our experience is that many of the existing programming languages for implementing MASs, such as IMPACT, 3APL, CLAIM, Jason, and Jadex [7], are based on techniques and approaches from computational logic. In these programming languages, computational logic techniques are used to model various mental attitudes of agents such as beliefs and goals, planning components, and reasoning components.

References

1. http://www.inderscience.com/browse/index.php.
2. http://www.agentgroup.unimore.it/aose05.
3. http://www.sics.se/tac.
4. http://www.agentcities.org/EUNET/Competition.
5. http://www.cs.uu.nl/ProMAS.
6. http://www.cs.uu.nl/ mehdi/al3promas.html.
7. R. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni. *Multi-Agent Programming: Languages, Platforms, and Applications*. Kluwer Academic Publishers, 2005.
8. R. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni. *Programming Multi-Agent Systems*, volume 3346. LNAI, Springer Verlag, 2005.
9. J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven information systems engineering: the TROPOS project. *Information Systems*, 27:365–389, 2002.
10. M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni. *Programming Multi-Agent Systems*, volume 3067. LNAI, Springer Verlag, 2004.
11. L. Padgham and M. Winikoff. Prometheus: A methodology for developing intelligent agents. In *Agent-Oriented Software Engineering III: Third International Workshop (AOSE'02)*. Springer, LNAI 2585, 2003.
12. F. Zambonelli, N. R. Jennings, and M. Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3):317–370, 2003.

Using Pheromones, Broadcasting and Negotiation for Agent Gathering Tasks

Simon Coffey and Dorian Gaertner

Imperial College London, SW7 2AZ, United Kingdom

Abstract. We describe an implementation of distributed, multi-threaded BDI-style [RG95] agents cooperating efficiently in a food-collecting scenario. Using ant-style pheromone trails and a pseudo-random walk procedure they explore the world uniformly, and negotiate to allocate collection tasks. Global information is disseminated via a publish/subscribe mechanism. The system is implemented using the concurrent logic programming language Qu-Prolog.

1 Problem description

For brevity's sake, a full description of the problem is omitted here. However, in addition to the constraints given in the competition specification, we have made two further assumptions: that the agents can move only in the directions North, South, East and West (i.e. diagonal moves are excluded), and that each agent can only carry one item of food at a time.

2 Design

A multi-agent system is typically characterised by the *distributed execution* of *communicative agents* that are *situated* in an environment.

We decided to use multi-threaded, logic-based, autonomous pseudo-BDI agents that are situated in an environment without central control. The environment process seeds food into the world, maintains the pheromone trails[1], sends percepts to the agents when requested and interfaces with the GUI.

2.1 Architecture

We designed our agents using an architecture loosely based on Rao and Georgeff's Beliefs-Desires-Intentions (BDI) model [RG95]. Each agent has *beliefs* about the state of the world including the location of food and the depot as well as beliefs about *claims* other agents have made. When an agent claims a certain piece of food, he informs the other agents about his *intention* to pick it up. Delivery of food and searching are other examples of intentions. *Desires* in our implementation are largely implicit, being limited to the built-in aim of each agent to collect and deliver as much food as possible in the shortest number of moves.

Each agent consists of two primary threads and a dynamic database (figure 2(a)). The knowledge thread receives percepts from the environment (*sensing*), updates the belief store depending on how it perceived the world, re-evaluates the intentions of the agent and communicates with other agents announcing certain events. The action selection thread then uses the current beliefs and intentions to decide which action to execute next. It informs the environment about its choice of action, which updates the world state and sends new percepts to the agent's knowledge thread.

[1] described in Section 2.8

2.2 Agent Language

Actions, percepts, beliefs and intentions are all sets of Prolog terms:

```
Action ::= [pickup, putout, move(Direction)]
```

where Direction is a variable representing north, south, east or west

```
Percept ::= [depot_same_cell, food_same_cell, has_food, has_moved,
             north(N), south(S), east(E), west(W)]
```

where N, S, E, and W are variables that represent cart, wall or a pheromone level

```
Belief ::= [at(X,Y), depot_at(X,Y), have_food, intends(Agent,
            Intention)]

Intention ::= [collect_food(X,Y), deliver_food]
```

where X and Y represent coordinates. Note that searching is never explicitly intended by the agent, but used as a default behaviour if no other intentions exist.

2.3 Action Selection

Agents choose their actions using *teleo-reactive (TR) programs* [Nil94], consisting of a priority-ordered sequence of condition/action rules. A simplified version of the TR program used is shown in Figure 1. This approach is particularly useful in scenarios like ours, in which durative behaviours (e.g. *explore*) are desired. It is important to note that at each percept/reaction cycle, the action chosen is only ever a single atomic one, belonging to the agent's set of allowed actions (defined above). For example, while `walk_to(X,Y)` appears to be a multi-step plan, it is in fact simply a set of rules which choose the agent's next atomic action; it must be repeatedly invoked in order to arrive at `(X,Y)`. Thus, the right-hand side of the rules in figure 1 are all either atomic actions, or programs which return an atomic action.

```
intends(deliver_food) ∧ believes(agent_at(depot)) → putout
  intends(deliver_food) ∧ believes(depot_at(X,Y)) → walk_to(X,Y)
                            intends(deliver_food) → explore
                               believes(at_food) → pickup
                       intends(collect_food(X,Y)) → walk_to(X,Y)
                                                ⊤ → explore
```

Fig. 1. Simplified action-selection TR program

Note that the action selection program does not manipulate beliefs, alter the intentions of the agent or handle negotiation in any sense; it operates solely on the current intentions and beliefs of the agent, returning only an action. All agent state manipulation is performed by the intentions thread (described in Section 2.4), which runs in parallel to the action thread, ensuring a consistent set of beliefs and intentions for the action selection program to use.

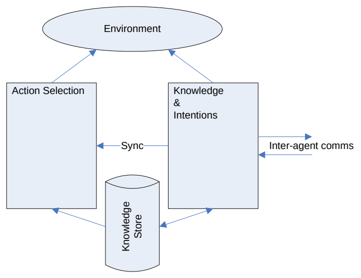
2.4 Intention Selection & Knowledge Maintenance

The intention selection thread takes the form of a message-processing cycle. While awaiting the next set of percepts from the environment, it listens for broadcast messages and negotiation requests from other agents, updating its beliefs and intentions accordingly. This is the only place in which modification of the agent's `believes(...)` and `intends(...)` dynamic predicates is permitted. For example, if agent `red` receives a broadcast message informing him that agent `blue` is claiming food at location (5,9), it will add the term `believes(intends(blue, collect_food(5,9)))` to its dynamic knowledge base.

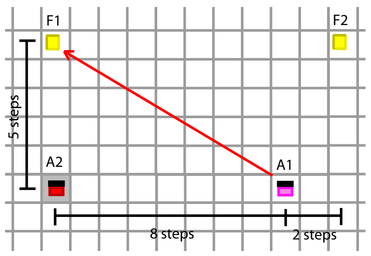
When a set of percepts is received, the agent first updates its beliefs about the world state using the new percepts. Since the set of percepts it can receive is relatively limited, this is achieved with an explicit set of handling routines for each type of percept. It then decides whether to send any negotiation requests, and finally re-evaluates its intentions accordingly. It does so using a series of declarative conditions, made possible by the backtracking operation of Prolog-style languages. For example, the delivery cost function for a particular item of food is simply written with two rules,

```
cost_of(food(X,Y),Cost) :-
    believes(agent_at(AgX,AgY)),
    believes(depot_at(DepX,DepY)),
    manhattan(AgX,AgY,X,Y,C1),
    manhattan(X,Y,DepX,DepY,C2),
    Cost is C1 + C2.
cost_of(food(X,Y),Cost) :-
    believes(agent_at(AgX,AgY)),
    manhattan(AgX,AgY,X,Y,Cost).
```

where `manhattan(X1,Y1,X2,Y2,D)` gives the manhattan distance between two points. This cost function is then called to find the optimum choice of food at the start of each turn (assuming there is any known food). If this food is believed to be claimed by another agent, negotiations are initiated with that agent. If the negotiation is unsuccessful, the agent will claim the cheapest unclaimed food (or retain whichever food it had previously claimed). Every new claim is broadcast to the other agents, enabling them to contact the "owner" of any food they wish to claim for themselves.



(a)



(b)

Fig. 2. (a) Negotiation example and (b) architecture design

2.5 Communication

Communication between agents utilises two of the main communication paradigms: publish/subscribe, and point-to-point messaging. The former is used for global knowledge sharing, while the latter is used for efficient negotiation between specific agents, as well as agent/environment communication. Each agent places a *subscription* for messages about food/depot locations and agent commitments at a remote server. When one agent finds an item, or claims some known food, it will *publish* a notification about this event to the server which in turn will inform all other agents that have subscribed to this event. This allows for dynamic addition of new agents to the scenario without having to change the running system. Negotiation is achieved using asynchronous message-passing. This is more efficient than using the broadcast system, since negotiation is always bilateral in our implementation; there is no need for all agents to be party to the negotiation messages.

2.6 Negotiation

In order to most efficiently allocate the collection of known food to each agent, we allow our agents to negotiate over the targets of their intentions. The agents have a defined policy only with respect to individual negotiations, namely to minimise the combined cost of delivery for the two negotiating agents. This is achieved by examining each agent's next-best option, and optimising accordingly. The implicit global effect of this policy is to minimise the total delivery cost of all known deliveries. This is the result of a series of bilateral negotiations; no single agent takes responsibility for optimising the entire set of deliveries.

Figure 2(b) illustrates an example where negotiation can improve the efficiency of food collection. It shows a snapshot of the environment state, in which agent A_2 has just delivered some food, agent A_1 has claimed and intends to pick up food F_1 and some other agent that was already carrying food accidentally discovered and broadcasted the existence of food F_2. Without negotiation, A_2 would claim F_2 and collect and deliver it in 30 steps while A_1 drops off F_1 in 18 steps.

Note that A_1 would not volunteer to pick up F_2 since this would increase his personal delivery cost to 22 steps. We therefore allow A_2 to send a bid to A_1, requesting permission to collect F_1 instead. It sends its cost of collecting F_1, plus the delivery cost of its next-best option (in this case F_2). A_1 will then consider the request, ceding responsibility for F_1 if the total delivery cost after the swap is reduced. The re-allocation allows agent A_2 to pick up and deliver F_1 in 10 steps at the expense of a small increase in the other agent's delivery cost. In terms of welfare economics, both the egalitarian and the utilitarian social welfare is improved[2].

2.7 Agent Roles

After initial experimentation with all agents performing as described above (i.e. searching until they first find food, then immediately delivering it), it became obvious that except in the most food-rich environments, knowledge about food locations was almost non-existent. The agents thus simply randomly walk until they first find food, which they immediately pick up and deliver, giving no opportunity for task optimisation. We therefore implemented a second type of agent, a scout. Upon finding food, a scout will not pick it up, but will merely broadcast its location to the gatherer agents, and continue searching. This can be viewed as a second implicit desire, with the scout agent's desire being to gather information rather than food.

[2] We take utility to be the negated cost of delivery, so that a shorter delivery yields a higher utility. The cost of delivery is the number of steps from the agent's location via the location of the food to the depot location.

As implemented, the scout is statically determined; it may not switch to delivery mode. However, it is easy to envisage a scheme in which agents switch to scouting dynamically, thus completing the full BDI repertoire of mental attitudes. In a scenario in which all the agents are randomly searching, the first agent to happen upon food might switch to scout duty, combing the rest of the area while the other agents collect the food it has discovered. In this manner, unexplored areas (and hence concentrations of food) would be explored, rather than the first food simply being delivered and forgotten about.

2.8 Exploration

Initially the agents do not know where the depot is located or which cells of the grid contain food. They must therefore explore the world around them. Dividing the world into quadrants and assigning each agent to a quadrant would be the most efficient way to explore the world completely, but this ignores over-exploration of repeatedly visited areas (i.e. the area around the depot). For this reason, a pseudo-random walk technique is used for exploration, utilising trail markers to ensure that agents prefer to explore cells that have not been as frequently visited.

A completely random walk based on Brownian motion would not be efficient enough since it tends to over-explore some areas at the expense of others. We chose to implement a more directed approach based on pheromones. Each agent drops a fixed amount of pheromone each time it enters a cell, similar to the methods used in ant colony optimisation ([DMC96], [GC05]). An agent can *smell* the concentration of pheromone in its neighbouring cells and probabilistically decides to move in a direction which is under-explored. If there are one or more unexplored adjacent cells, it will always choose a move to one of these cells. This pseudo-random walking leads to the uniform exploration of the world. It also compensates for over-exploration of the area around the depot; the repeated trips of agents to this location mean it would be heavily over-explored if a quadrant-based strategy were used.

However, there is a disadvantage to simply counting all the visits to a cell since the start of the simulation. In an environment in which food is continuously appearing, the goal of a search algorithm must be to ensure each cell is visited as regularly as possible. In a system with permanent pheromones, a cell that has been visited only once, but very recently, appears more attractive to explore than a cell that was visited 10 times, 100 turns ago. In fact, the opposite is true, and the cell with "stale" pheromones should be explored preferentially. For this reason, a pheromone decay mechanism has been implemented and proved useful, whereby pheromone values decrease over time according to a variety of formulae. This ensures that cells which were over-explored in the past do not get unreasonably ignored.

A further advantage of random walking is its use in avoidance of deadlocks. When two agents block each other's paths they will randomly move out of the way. While they may not successfully avoid each other instantly, due to the random choice of direction, the avoidance routine inevitably resolves the deadlock, since it is statistically impossible for both agents to choose the same move for ever. In addition, this method is much simpler to implement than exhaustive characterisation of every possible deadlock, along with explicit strategies for resolving them.

Most importantly, using this flexible movement behaviour our implementation adapts very easily to unknown and even dynamically changing environments.

3 Implementation

Our design requires an implementation language that allows for multi-threaded execution of agents. We chose *Qu-Prolog* [RW03] because it allows for easy, declarative description of the higher-level reasoning involved in intention selection and negotiation. Its flexible system of dynamic predicate manipulation also provides an unconstricted environment in which to construct and modify the simple agent language we have used, while simultaneously being descriptive enough to allow the lower level algorithms to be concisely expressed.

The publish/subscribe mechanism we described in Section 2.5 is realised using broadcasting via an *Elvin* [SAB^{+}00] server. Direct negotiation between agents makes use of the Interagent Communication Model (ICM). Its communication server provides agent naming facilities and the means to encode, transport and queue symbolic messages.

Effectively, we are using three forms of communication—point-to-point communication for negotiations, broadcasting for events and knowledge sharing, and indirect communication via the environment using pheromones for exploration.

4 Analysis/Conclusion

The foundation of the broadcast and negotiation techniques implemented is a good supply of information about the environment. In scenarios where there is more known food than the agents can collect at once, these techniques have a potential to improve the utilisation of the agents, since the time spent conducting relatively unguided searches is limited. However, this knowledge of food locations needs first to be obtained, hence the introduction of a scout agent. The impact of the various techniques implemented is briefly assessed here.

For our quantitative analysis, we fixed the depot at location (10,10) and ran the simulation until 100 items of food had been collected and delivered. The agents were still required to discover the depot on each run. Ten runs of the simulation were conducted for each scenario. The average number of steps for one food delivery (μ_{steps}), the standard deviation of the number of steps (σ_{steps}) and the average number of successful negotiations per delivered food item (μ_{neg}) have been measured.

Table 1. Quantitative results when food is seeded every 20 seconds

Scenario	μ_{steps}	σ_{steps}	μ_{neg}
4 gatherers (no pheromone, no negotiation)	73.7	3.32	n/a
4 gatherers (pheromones, but no negotiation)	55.7	1.27	n/a
4 gatherers (pheromones and negotiation)	56.4	2.07	0.20
3 gatherers and 1 scout (pher. and neg.)	55.6	1.1	0.29

This shows that adding guidance to the randomly walking ants with the help of pheromones significantly improves their behaviour. However, adding negotiation does not seem to improve the results. We believe this is due to the low rate at which food is seeded into the environment. The negotiation usually improves a combined delivery of two ants by about 10%. However, the ants spent the majority of their time searching for food or the depot. Only about 10% of their time is spent collecting and delivering and so the improvement achieved by adding negotiation is only 10% of 10%, equivalent to 1% overall.

In the high seeding-rate environment (with food seeded every 7 seconds), having a dedicated scout agent proves significantly detrimental to the team performance. This is unsurprising, as with high rates of seeding, food is sufficiently abundant that the agents have no trouble finding food on their random walk. The team with a scout still performed better than would be expected of a team consisting solely of three gatherers, however, experiencing only a 9% performance drop despite effectively losing a quarter of the delivery capability.

Table 2. Scout effect with varying seeding rates

Scenario	Seeding	μ_{steps}	σ_{steps}	μ_{neg}
4 gatherers	high	28.9	0.74	0.22
3 gatherers and 1 scout	high	31.6	1.28	0.18
4 gatherers	low	56.4	2.07	0.20
3 gatherers and 1 scout	low	55.6	1.1	0.29

In the low seeding-rate environment (with food seeded every 20 seconds), however, the benefit of the scout agent completely compensated for the loss of delivery capacity, roughly equalling the delivery rate of four gatherers. In effect, the scout provides sufficient global knowledge to allow the agents to employ their high-level reasoning much more frequently, resulting in more efficient collection of the known food. As the delivery agents leave trails every time they visit the depot, the scout agent tends to explore the areas further from the depot, discovering concentrations of food that the delivery agents are unlikely to find.

As predicted, the usefulness of the scout agent inevitably depends on the rate of seeding. In an environment with a high rate of seeding, the food density is such that it becomes more efficient to simply have all agents collecting, since they are likely to find food soon after leaving the depot on their random search. Additionally, a scout agent in this situation tends to over-explore the edges, drawing the gatherer agents further from the depot than necessary. In a food-sparse (and thus information-poor) environment, however, the scout becomes more useful despite the loss of one agent's delivery capacity.

While our preliminary results do not show a scenario in which a scout agent provides a decisive advantage, they do indicate that there are situations in which a definite benefit exists. As a future avenue of investigation, we believe that allowing the agents to dynamically switch roles offers potentially superior results.

5 Acknowledgements

We would like to express our deepest gratitude to Silvana Zappacosta for allowing us to use and helping us to modify her visualisation written in Tcl/Tk. Furthermore, we are indebted to Keith Clark for encouraging us to participate in this contest and for many fruitful discussions.

References

[DMC96] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1):29–41, 1996.

[GC05] Dorian Gaertner and Keith Clark. On optimal parameters for ant colony optimization algorithms. *Proceedings of the International Conference on Artificial Intelligence 2005 (to appear)*, 2005.

[Nil94] Nils J. Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158, 1994.

[RG95] A. Rao and M. Georgeff. BDI agents: From theory to practice. *Proceedings of the 1st International Conference on Multi-Agents Systems*, pages 312–319, 1995.

[RW03] Peter J. Robinson and Michael J. Walters. *Qu-Prolog 6.3 Reference Manual.* The University of Queensland, 2003.

[SAB$^+$00] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content based routing with Elvin4. In *Proceedings of AUUG2K*, June 2000.

Reactive food gathering. CLIMA VI contest submission

Robert Logie[1], Jon G. Hall[2], and Kevin G. Waugh[2]

[1] Osaka Gakuin University, Faculty of Computer Science
2-36-1 Minami Kishibe, Suita Shi Osaka, 564-8511 Japan
`rob@utc.osaka-gu.ac.jp`

[2] The Open University, Department of Computing, Faculty of Mathematics and Computing, Walton Hall, Milton Keynes MK7 6BJ England

Abstract. This short paper describes the design and development of a simple agent system aimed at addressing the food gathering problem set for the 2005 CLIMA contest. Our system is implemented as a collection of reactive agents which dynamically switch between a number of behaviours depending on interaction with their environment. Our agents maintain no internal representation of their environment and operate purely in response to their immediate surroundings. The agents collectively map the environment co-operating indirectly via environmental markers and they use these markers to assist them in locating the depot when they discover food. The required behaviour emerges from the interaction between agents and the marked environment.

The application can be downloaded from:
`http://219.1.164.219/~robert/pwBlog/wp-content/CLIMAbuild.zip`

1 Introduction

Jennings et al. [1] note that a major selling point of purely reactive agent systems is that overall behaviour emerges from interactions between component behaviours and the agent's environment. This inherent simplicity makes reactive agents attractive but it also masks a number of difficulties. The most notable are those of designing agents in such a way that they can take account of non local information and in such a way as to be able to improve their agent level performance over time. Jennings et al. further note that agents using a large number of behaviours can quickly become too complex to understand.

Recent research in normative systems and, particularly, normative reactive systems may provide the means of describing and constraining agent behaviour in a manner which allows us to address this difficulty. For many problems in a tightly bounded environment – problems such as industrial process control or safety systems – reactive agents may be ideal and a fuller understanding of their potential behaviour will be beneficial in allowing their use in increasingly complex scenarios. This contest environment provides such an environment and is, we feel, ideal for the application of reactive agents.

Reactive agents generally operate by having predetermined behaviours or sequences of actions intended to deal with the various circumstances that the agent may encounter. As circumstances change an agent may switch behaviours. Our agent design involved identifying problems within the environment and designing behaviours to address them. This switching between behaviours brings a number of constraints. If an agent's behaviour involves maintaining a record of data and the agent switches to another behaviour, that does not maintain this data, then this data may become outdated. In a dynamic environment such internal world data may be dangerously out of date when the agent returns to using it and maintaining the data my be expensive for a resource bounded agent concentrating on other tasks. We avoid such problems by letting agents use only very local data and data about their internal state or history.

2 The problem — a general approach

We approached the problem by identifying sub-problems which we could associate with agent roles. The roles assigned were those of locating food and, when food has been found, transporting it to the depot. Both of these roles involve searching the environment, the former for food which may be at random locations and the latter the depot which remains in a fixed location. Clearly it is in the system's benefit to have all agents aware of the depot so when one agent finds it some means of indicating its presence to others will be a valuable asset. We have limited communications options by restricting our design to being purely reactive and we limit each agent to being able to carry only one food unit.

We have assumed that the depot location is unknown initially but that it remains in a fixed position throughout a run, when an agent discovers the depot it discovers its permanent location. This leads to a minor difficulty, our agents operate using only very local data and don't know their absolute position in the world [3] which means that they cannot remember an exact depot location. We address this by letting agents leave local markers in the environment. Agents use a random walk to search for the depot. When an agent finds the depot it initialises a "dropper" which allows it to leave a trail of pheromone like weightings on the cells that it visits after having been on the depot cell. Despite the extreme simplicity of this system it allows agents acting only with local data to co-operate in mapping their environment in a way which facilitates the task of carrying food to the depot.

The depot location problem within food transport role is addressed by three agent behaviours; *depot-searching*, the *depot-marking* and the *depot-seeking*. These behaviours, respectively, involve a random walk looking for the depot cell, a random marker laying walk searching for food and a directed pheromone gradient following walk whilst carrying food back to the depot.

[3] Co-ordinate values are only used as a means of keeping agents in bounds and displaying user friendly data. Beyond ensuring that the agent doesn't try to move out of bounds they are not used in any of the agent's operating decisions.

The second search problem – that of finding food – cannot be approached in the same way since food is deposited randomly by the system. Such random placement precludes structured search behaviour by the agents. Food may well appear in a location already searched. The specification indicates that food can only be seen on a cell that an agent is visiting so this rules out giving food a smell that agents can detect. We address this problem with a simple random walk and there is one *food-searcher* behaviour assigned to the food locating role. This behaviour may be concurrent with the depot-searcher and depot-marker behaviours.

Searcher behaviours involve random walks and seeker behaviours involve trail following. There is no food-seeker behaviour so food searching is always a random process and its performance will not improve over time. Seeker behaviours involve using environmental markers left by agents to track previously located objects with persistent locations. It is expected that the performance of seeker type behaviours will improve over time as the environment is more accurately mapped during the random walk of food-searchers.

The food searcher and depot searcher behaviours can operate concurrently. Considering the agent's behaviour in this manner provides a convenient method for analysing behaviour transitions, these are shown in figure 1.

Bonabeau et al. [2, page 26] describe a broadly similar process where ants influence or *recruit* other ants so as to guide them towards persistent food sources, such recruitment based solely on pheromones is known as *mass recruitment*. The depot-marking behaviour is an instance of this mass recruitment as depot-searchers (agents that have yet to find the depot) make use of the pheromone trails from agents that have already located the depot. We have also briefly experimented with other environmental marking methods but felt that these were uncomfortably close to requiring global knowledge or data, something which we are trying to avoid.

3 The agents

Our agents have two modules, a simple reactive core and a move manager. The reactive core senses details of the agent's immediate environment and takes actions depending on its percepts. The agent's roles and component behaviours have been briefly described in section 2. The agent's "cycle" involves sensing its environment, selecting a behaviour, executing that behaviour then making either a directed or random move. Behaviour selection is very dynamic and an agent may switch behaviours on each cycle through its core module. In this environment all behaviours either execute concurrently (such as food-searcher and depot-searcher), or one is suppressed by historic actions (such as having located and picked up food the food-searcher behaviour is suppressed in favour of the depot-searcher behaviour).

Our agent's pheromone tracking behaviour is very simple, finding the depot triggers the agent's depot-marking behaviour causing the agent to prime its trail marker and reinforce any environmental markers in locations it passes through. It

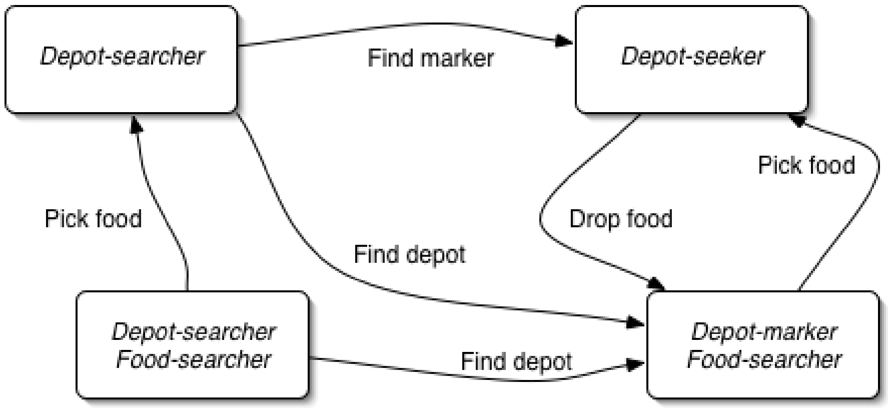


Fig. 1. Agent behaviour transitions.

will reinforce other agent's markers but does not reinforce its own. Finding food will trigger the depot-seeker behaviour causing the agent to stop marking and try to get back to the depot by following marker gradients. The other environment marking methods, mentioned in section 2 that we briefly experimented with were of comparable complexity.

The move manager is coupled to agent core and simply makes sure that the agent doesn't move out of the world's boundaries. This coupling is loose in the sense that the agent doesn't monitor what the move manager does and merely requests a pheromone gradient directed move or a random move. Non determinism caused by the move manager not executing a directed move is handled by the agent's operating in cycles, each cycle is a sense, select, act sequence. This small source of non determinism is probably swamped by the non deterministic aspects of food location in the environment.

4 The problem — logical aspects

Agents *ought* to take food to the depot and they *ought* to do this in as efficient a manner as possible. Considering what agents ought to do allows us to adopt a deontic view of the system but this brings difficulties. Horty notes[3, page 36] that standard deontic logic partitions future worlds into sets of ideal worlds and non ideal worlds. Agents either take food to the depot or they don't, there's no notion of a good or bad way of doing this and, consequently, no notion of improving performance. Norms are typically a social phenomena [4] which makes them intrinsically a multi agent concept but do they have a place in our system? Boella and van der Torre [5] indicate that an important feature of norms is that they allow for behaviour that deviates from ideal and this may allow us to consider norms as a performance improving influence. Our agents are extremely simple, they have fixed transitions between behaviours (see figure reffig:transitions) and no internal systems to allow considered choice, their operation appears to be constrained rather than norm governed. Agents are constrained by their behaviour to drop food on the depot but they may or may not take the best route from where the food was located. Our agents are always capable of taking food back to the depot, a random walk either way would, in this bounded environment a random walk would eventually locate the depot.

When every cell in the world has been marked the agents use a subset of their available behaviours, the depot-searcher is no longer required and transitions are only between depot-seeker and depot-marker/food-searcher behaviours. We think that this can be described as an emergent norm which guides agent behaviour away from the depot-searcher behaviour. If our belief holds and we consider our system as a meta agent then this may be a very simple and possibly degenerate example of of what Boella and van der Torre describe in [6]. Our simple agents delegate the task of improving their performance to an emergent norm and they contribute to its emergence.

5 Observations

Our system performance evaluation had two criteria, the directness of the route taken by agents carrying food back to the depot and whether or not food accumulated in the environment. Test runs were carried out by seeding the environment with a few food units then starting the agents. Initial agent performance is rather poor, agents rely on random searches for both the depot and food. Agents that have found food wander at random and don't appear to be doing anything useful whilst food continues to appear. When one agent finds the depot and begins marking the environment other agents gradually move from random depot-searching to pheromone gradient following depot-seeking. At first this means following, in reverse, another agent's random walk so as to reach the depot. Over a period of time the gradient mapping spreads more widely and agents begin taking more direct routes to the depot with a concomitant performance improvement.

Agents will occasionally becoming trapped by a "livelock". This livelocking manifests itself when an agent appears to walk repeatedly over the same looped path. This only occurs when a food carrying agent is following a pheromone gradient and encounters local maxima. Because the agent follows gradients without backtracking these local maxima may trap the agent. Livelock may be broken by another agent passing through a cell adjacent to the loop and altering pheromone levels sufficiently to allow the trapped agent to escape. The competition system has a small number of agents and if there is a high food density then there is a risk that all four will become livelocked especially where local maxima form within a few grid squares of the depot, a location to which depot-seeker agents are already drawn.

Intuitively if there are non food carrying agents then there is a chance of a livelock being broken. Dealing with this difficulty whilst maintaining a local data only approach is an interesting problem.

6 Installing and running the application

The application can be downloaded from the link in the abstract. This is a zip file which contains seven files, one executable, four DLLs which define agent behaviour, a system configuration file indicating which agents to automatically load on startup and a PDF with brief user instructions. Copy all of these files into a directory on the target machine and run the executable. The program will automatically load the agent DLLs specified in the configuration files and is ready to run.

7 Next steps.

Despite their simplicity our agents do, what we consider, a good job at carrying food to the depot and improving their performance over time. We have concentrated on the depot and not paid much attention to food location simply leaving this to an unstructured, random search. Dealing with the occasional appearance of livelock whilst maintaining a local data only approach presents an interesting problem. Adopting a normative approach we could prohibit livelock. Saying that agents ought not to livelock implies an avoidance capability. One approach is to have "defender agents" [7] which look for possible livelocks and release trapped agents. The difficulty of doing this using only local data is obvious. Our agents are very simple but considering them as a normative system gives a rich view of their interactions and raises a number of questions about how best to improve their performance. If an agent finds food and is unable to pick it up then marking that food location – in a similar manner to the depot – may intuitively seem to be a good step but this may lead to a greater possibility of all agents becoming livelocked.

Our system was developed solely for the CLIMA contest but it has opened up a number of interesting areas to investigate. The observations [1] in section 1 seem to hold even for this very simple system. A normative approach may provide a means of better understanding the interactions in reactive systems.

References

1. Jennings, N., Sycara, K., Wooldridge, M.: A roadmap of agent research and development. Autonomous Agents and Multi-Agent Systems **1** (1998) 7–38
2. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence - from nature to artificial systems. Santa Fe institute studies in the sciences of complexity. OUP (1999)
3. Horty, J.: Agency and deontic logic. OUP (2001)

4. Conte, R., Castelfranchi, C.: Cognitive and social action. UCL press (1995)
5. Boella, G., van der Torre, L.W.N.: Fulfilling or violating obligations in normative multiagent systems. In: IAT, IEEE Computer Society (2004) 483–486
6. Boella, G., van der Torre, L.W.N.: Attributing mental attitudes to normative systems. In: AAMAS, ACM (2003) 942–943
7. Boella, G., van der Torre, L.W.N.: Norm governed multiagent systems: The delegation of control to autonomous agents. In: IAT, IEEE Computer Society (2003) 329–335

Extending Tropos for a Prolog Implementation: A Case Study Using the Food Collecting Agent Problem

Carlos Cares[1,2], Xavier Franch[1] and Enric Mayol [1]

[1] Dept. Llenguatges i Sistemes Informàtics - Universitat Politècnica de Catalunya,
Jordi Girona, 1-3 08034 Ph. : 43-93 413 7839,
Barcelona, Spain.
`{ccares,franch,mayol}@lsi.upc.edu`

[2] Dept. Ingeniería de Sistemas, Universidad de La Frontera,
Av. Francisco Salazar 01145, Casilla 54-D, Ph 56-45 325000
Temuco, Chile.

Abstract. There is a recognized lack of Agent Oriented Methodologies to translate a detailed design to a software implementation; here we address this problem with a solution approach. *Tropos* is one of the most used methodologies to design agent systems and we use it to show a design for the Food Collecting Agent Problem. Our solution includes autonomous behaviour, beliefs, multiple roles playing, communication and cooperation in a simple way. We propose a method to generate a Prolog implementation from a *Tropos* detailed design, adding a step allowing relevant decisions being incorporated at design time. Besides we show how to get the Prolog implementation from this detailed design. Our experience shows that this proposal is an intuitive, direct and effective way to get a Prolog implementation for an agent system. We end the paper with illustrations about our collecting team in action.

1 Introduction

Nowadays there is a recognized lack of Agent Oriented Methodologies (AOM) at the implementation stage [1, 2]. *Tropos* [3, 4] is one of the most used AOM, however, in spite of all its virtues, its implementation stage does not have enough guidance for declarative software implementations [5]. In this paper we address this situation and propose a method to get a Prolog implementation starting from a *Tropos* design. To illustrate our method we use the Food Collecting Agent Problem (FCAP) as a case study, which is about a grid-like environment where agents can move from one slot to a neighboring slot if there is no agent already in the target slot. In this world food can appear in a randomly way in an empty slot. There is a special slot where the agents must collect the food, named the depot. In the next section we show briefly the stages of *Tropos* and our design for the FCAP, in section 3, an additional design step oriented to get a Prolog implementation is proposed and finally we show how to convert this output into a computer program.

2 Using *Tropos* for the FCAP

Tropos [3, 4] is an agent-oriented methodology for building software systems. It is adequated to describe both the social (organizational) environment and the system itself. According to [5], *Tropos* covers from early requirements to implementation with a different clear focus on each stage: (1) Early requirements focus on social context; (2) Late Requirements focus on system-to-be; (3) Architectural Design, focus on systems components; (4) Detailed Design and (5) Implementation, both focus on software agents.

Tropos uses the concepts of actors, which can be organizational, human or software; positions, roles and agents, as specializations of actors; goals and social dependencies for representing the commitments or agreements of actors (*dependees*) to other actors (*dependers*). The type of the dependency depends on the intermediary element (*dependum*) between actors. It can be *goal* (hard or soft), *plan* or *resource*. Thus the basic structure of social representation is the *dependee-dependum-depender* relationship. In the figure 1 and in the figure 2 we have illustrated the graphical representation of *Tropos* constructors according to their use. For further details about *Tropos* see [5].

In *Tropos*, at the Early Requirements stage, the analysis of the environment must be done. Since in the FCAP case study does not have a social context, we omit this stage.

At the Late Requirements stage the system-to-be is analyzed and the functional and non-functional requirements are specified. For FCAP we recognize two main actors, the food provider (FP) and the collector team (CT). There is a main dependency that represents the need of the CT from the FP for food, either initial or produced, but also the FP has constraints for the behavior of the CT in the food environment. Although there is an evolution of diagrams inside of this stage, we show the final output in the figure 1.

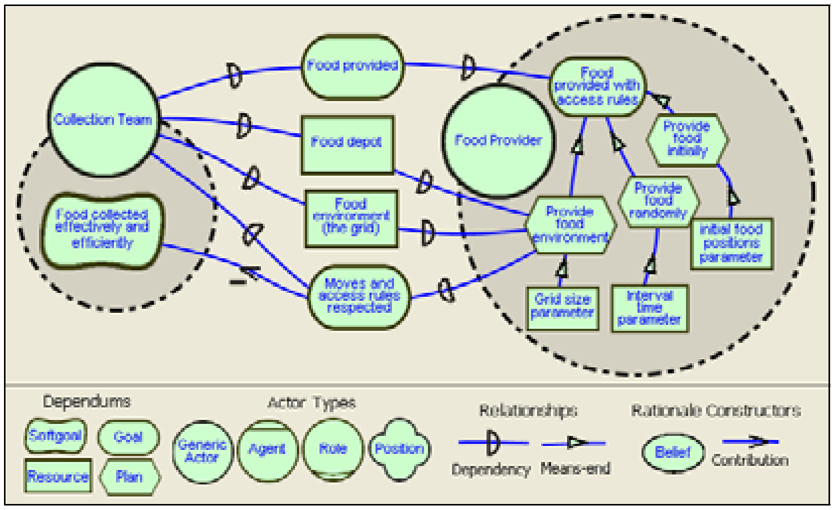


Fig.1 The output of the Late Requirements stage

In the Architectural Design stage the global architecture of the system is analyzed, new actors are incorporated and their main capabilities are identified. In our case we have added a *teamMember* position that represents all member of the team. Moreover we have decided to tackle the problem with the roles *collector* (for gathering food and disposing it in the depot) and *explorer* (for looking for food in the grid). Finally we have delegated in the *ruleGuard* role the goal to keep an adequate behavior. When we specify that the position *teamMember* covers the *ruleGuard* role, means that all members of the team must play this role. In figure 2 we show the output of the Architectural Design. For simplicity we have omitted the positive contributions from the *Team Member*, *Collector*, *Explorer* and *Rule Guard* goals to the main softgoal of the *Collection Team*.

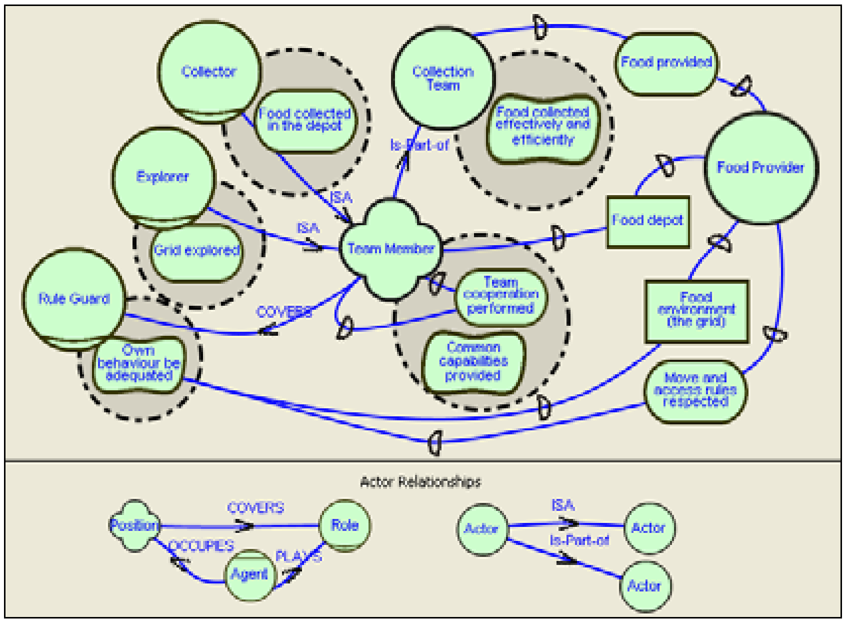

Collector
Food collected in the depot
Collection Team
Food collected effectively and efficiently
Food provided
Food Provider
Explorer
Grid explored
ISA
ISA
Is-Part-of
Team Member
Food depot
Rule Guard
Own behaviour be adequated
COVERS
Team cooperation performed
Common capabilities provided
Food environment (the grid)
Move and access rules respected
Actor Relationships
Position
COVERS
Role
OCCUPIES
PLAYS
Agent
Actor
ISA
Actor
Is-Part-of
Actor

Fig.2 The output of the Architectural Design stage

At the Detailed Design stage, each actor is individually analyzed and each goal of the Architectural Design is decomposed to specify the actor capabilities. Thus, for the *ruleGuard* role, we have designed capabilities to access the own position, to pick up food, to know empty neighbor slots and to move just into these slots. For the *teamMember* role we have identified a *belief* about the food environment, this *belief* can be updated with agents interaction, thus the cooperation among team is based on sharing their beliefs. Moreover we have provided direct access to the position of the depot and we have designed the capability to advance forward a target point in the grid. For the *collector* role we have the capabilities of disposing food in the depot, moving to the depot, moving for a guessed food position (based on its *belief*) and, in the case of no food information, looking for it in unvisited slots. We have designed a major strategic, which specify that collectors have different search spaces according the collectors quantity and the size of the food environment.

Finally for the *explorer* we have designed the capability of moving sequentially by the grid, and the necessary data resources to support this capability have been identified. In the figure 3 we show a partial view of the Detailed Design output, we illustrate this stage with the R*ule guard* and T*eam member* roles, for simplicity we have omitted the relationships and *softgoal* contributions among the actors. The simplicity of the *explorer* role, based on a sequential moving over the grid, allows focusing on less classical design aspects. In the next section we show our extension for the detailed design of the *collector* role.

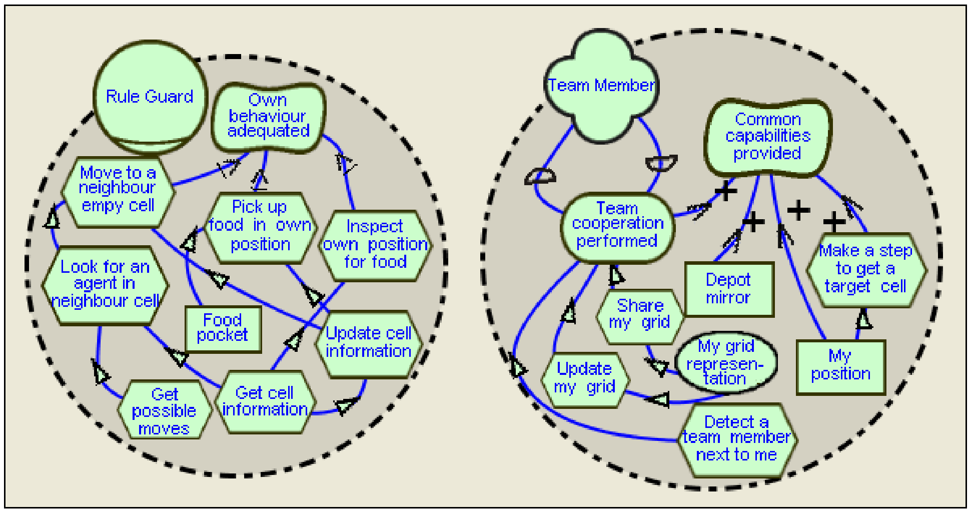

Rule Guard
Own behaviour adequated
Move to a neighbour empty cell
Pick up food in own position
Inspect own position for food
Look for an agent in neighbour cell
Food pocket
Update cell information
Get possible moves
Get cell information
Team Member
Common capabilities provided
Team cooperation performed
Depot mirror
Make a step to get a target cell
Share my grid
My grid represen-tation
Update my grid
My position
Detect a team member next to me

Fig.3 Detailed design for Rule Guard and Team Member roles

3 Extending Detailed Design for Prolog Implementation

In this section we explain how to extend the Detailed Design stage to get a declarative implementation. We propose to replace the AUML activity diagrams used in *Tropos* by scenario sequences. These have been used at the requirements stage in a web-based software system [6]. We think that it is a simple way to specify sequences when they are needed. In Prolog these sequences must have the convenient order to prove the logical goals. This has relevant logical and efficiency effects hence it is very convenient to have a representation of these decisions at design time. Besides we propose *goal* and *plan* root elements be implemented like Prolog. In the figure 4 we show a scenario sequence for the *collector* role, here the design indicates that the first goal to be proved is *put food in the depot*, but this means to check that the agent is over the depot and has food in the buffer, otherwise the second goal, *go to the depot*, should be proved, etc.

For *goal* and *plan* root elements we propose to specify the programming activation time, thus we propose four implementation attributes, namely *at begin*, *at end*, *at call* and *always*. *At call* means that the goal or plan is invocated from another goal or plan. *At begin* and *at end* indicate that the selected goal or plan should be proved just one time, at start of run time or at the end. *Always* means that the goal or plan should be permanently proved. If there is a multi thread Prolog implementation it is suggested that each goal or plan with the *always* attribute be a different thread.

Finally we propose to make the decisions about data representation. The design elements, which require having a data representation, are *resource* and *belief*. In Prolog we have two main choices: a data structure (generally a list) or the knowledge data base (KDB) included in Prolog. In our case, we have decided to represent the food environment with predicates (KDB) and the rest of *resource* and *belief* elements with lists.

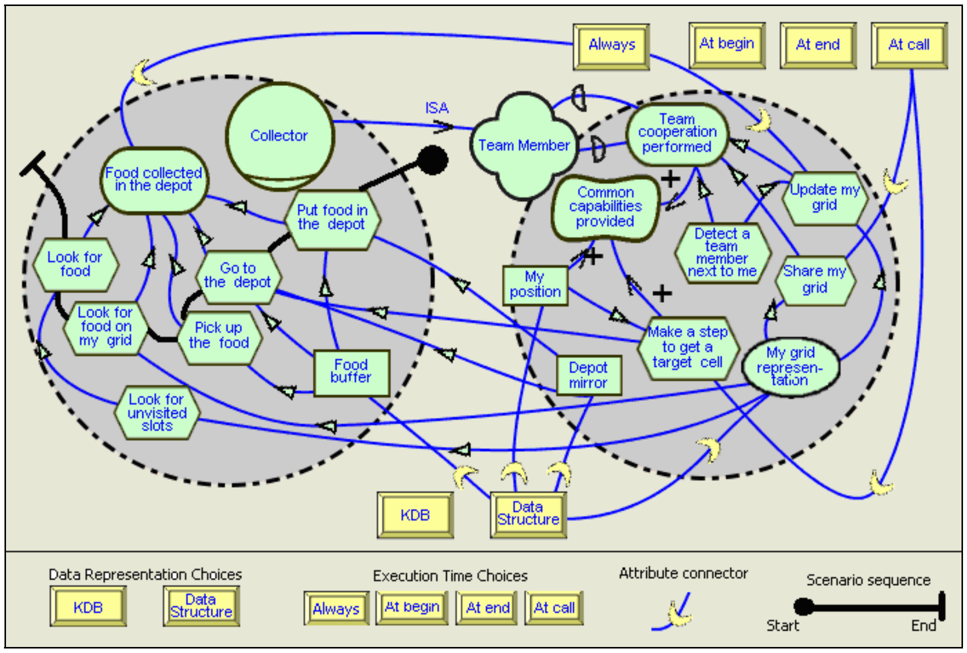
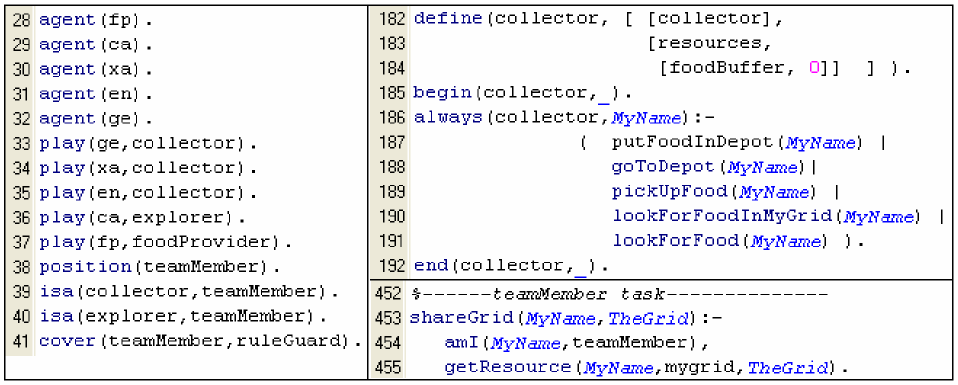


Fig.4 Partial Extended Detailed Design for FCAP

At implementation time we have a set of recommendations to translate the extended detailed design. First, make the instantiation of the problem with the predicates *agent*, *play*, *position*, *isa*, *cover*, etc., i.e. the actor types and actor relationships from *Tropos*. For FCAP we have generated a specific instantiation with five agents, a *Food Provider* agent (*fp*), an explorer agent (*ca*), and three collectors (*en*, *xa* and *ge*), we show this from lines 28 to 41 in figure 5. Second, to implement *resource* and *belief* elements as part of the *define* predicate, identifying the name of the role as first argument, and a data structure which define the resources (e.g. lines 182 to 184 in figure 5). Third, to group root elements (*goal* and *plan*) under the identified activation times, each activation time is a predicate that needs the actor type (atom) and the agent name (variable); this is illustrated from lines 185 to 192 in the figure 5. Fourth, to program the goals and a plan using a set of predicates that act over the defined data structures and clauses. This step motivates the reuse of already programmed predicates but is not an automatic step because the specific clauses depend on the semantic of the goal or plan. For example we show, in figure 5, the *amI* and *getResource* predicates that implement the *shareGrid* plan (lines 452 to 455).



```
28 agent(fp).
29 agent(ca).
30 agent(xa).
31 agent(en).
32 agent(ge).
33 play(ge,collector).
34 play(xa,collector).
35 play(en,collector).
36 play(ca,explorer).
37 play(fp,foodProvider).
38 position(teamMember).
39 isa(collector,teamMember).
40 isa(explorer,teamMember).
41 cover(teamMember,ruleGuard).
```

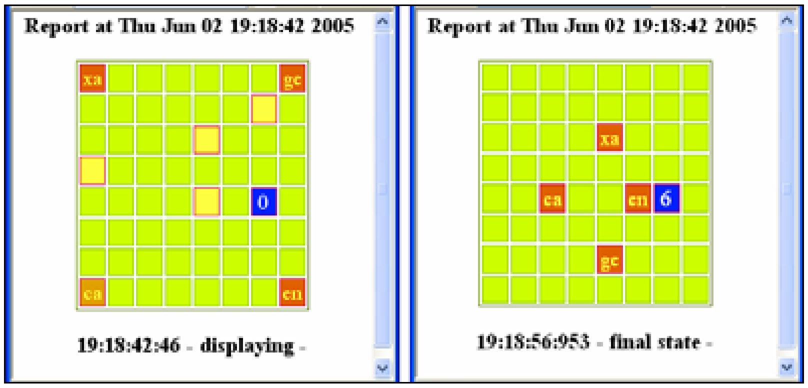
```
182 define(collector, [ [collector],
183                     [resources,
184                      [foodBuffer, 0]]   ] ).
185 begin(collector,_).
186 always(collector,MyName):-
187                 (  putFoodInDepot(MyName)  |
188                    goToDepot(MyName)|
189                    pickUpFood(MyName)  |
190                    lookForFoodInMyGrid(MyName)  |
191                    lookForFood(MyName)  ).
192 end(collector,_).
```

```
452 %------teamMember task--------------
453 shareGrid(MyName,TheGrid):-
454     amI(MyName,teamMember),
455     getResource(MyName,mygrid,TheGrid).
```

Fig.5. Direct implementation of the detailed design

And fifth, we have made the goals that allow the access to the set of *begin*, *end*, and *always* predicates, namely *runBegin*, *runEnd*, and *runAlways* (without arguments). It is very important to note that the definition order of agents will be the calling order, thus if we run *runBegin*, it will be executed first the *fp* begin and the last one will be the *ge* begin (according to lines 28 to 32 in figure 5). These logical goals are generic and could be used in any agent system implemented in Prolog.

The resulting system generates an output that we run under a web browser. We have set the system with an 8x8 food environment and fifteen seconds for a running. We illustrate the resulting system in figure 6 where de depot is the dark slot with the number 0 at start and 6 at the end. The food is into light slots and the agents are the slots with the strings *ca*, *xa*, *en* and *ge*.



Report at Thu Jun 02 19:18:42 2005

19:18:42:46 - displaying -

Report at Thu Jun 02 19:18:42 2005

19:18:56:953 - final state -

Fig. 6. Running the resulting agent system

5 Conclusions

In this paper we have proposed a specific way to get a Prolog implementation from a *Tropos* design. Although we have used the FCAP, the approach is a generic developing proposal based on the atomicity of the detailed design. However a set of different projects should be carried out to get stronger evidence about its utility. Moreover this approach requires a set of generic predicates which do not have a design representation in each specific problem. However, when they have been developed, the programming is intuitive and direct. Thus a relevant part of the code could be generated automatically and the rest could be sufficiently documented for programming aid.

About FCAP we have designed a solution using multiple roles playing (*Team member*, *Rule guard* plus *Collector* or *Explorer*). The agents cooperate in the solutions sharing information about their belief of the world. Besides, the common behavior is grouped in the *Team member* role, being an efficient solution to the problem. Our experience indicates that this proposal is an intuitive, effective and efficient way to implement agent-oriented systems.

Acknowledgement

This work has been done in the framework of the research project UPIC, ref. TIN2004-07461-C02-01, supported by the Spanish Ministerio de Ciencia y Tecnología. Carlos Cares wants to acknowledgment to MECE-SUP FRO0105 Project of the chilean government and the University of La Frontera, who supports his stand at Universitat Politècnica de Catalunya. Besides we want to thank to Francesca Toni and Paolo Torroni for the conceded grant to expose our report at CLIMA 2005.

References

1. Dastani M., Hulstijn J., Dignum F., Meyer J.:Issues in Multiagent Systems Development. Third International Conference AAMAS04, Columbia, USA, July (2004), 920-927.
2. Hoa, K. and Winikoff, M.: Comparing Agent-Oriented Methodologies, In the proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems, Melbourne, (at AAMAS03), July (2003).
3. Giorgini P.,Perini J., Mylopoulos J., Giunchiglia F, Bresciani P.:Agent-oriented software development: A case study. In Proceedings of the Thirteenth International Conference on Software Engineering – Knowledge Engineering (SEKE01), Buenos Aires (2001).
4. Perini A.,Brisciani P.,Giunchiglia P., Giorgini P., Mylopoulos J.: A knowledge Level Software Engineering Methodology for Agent Oriented Programming. In Proceedings of the Fifth International Conference on Autonomous Agents, Montreal, Canada, May (2001).
5. Sannicoló, F., Perini, A., Giunchiglia, F.: The Tropos modelling language. A User Guide, Technical report DIT-02-0061, University of Trento, February (2002).
6. Liu L. and Yu E.:Designing Web-Based Systems in Social Context: A Goal and Scenario Based Approach, Lecture Notes in Computer Science Vol 2348, Springer-Verlag, Berlin Heidelberg New York (2002), 37-51.

Strategies for Multi-Agent Coordination in a Grid World

Eder Mateus Nunes Gonçalves and Guilherme Bittencourt

UFSC - Federal University of Santa Catarina
DAS - Automation and Systems Department
88040-900 Florianópolis - SC - Brazil
{ eder | gb }@das.ufsc.br

Abstract. In this work, we describe strategies for multi-agent coordination, where adequate coordination means a system performance increase. In the main strategy, when a agent can't act by any reason, it choose the agent more capable in the environment to execute these action. The results show the strategy efficacy, especially when the environment increase the necessity for a reaction.

1 Introduction

The advantages obtained with a multi-agent approach can be easily lost if an adequate coordination process between agents can not be established. To explore the real possibilities of a multi-agent strategy, the agents in the society must be able to cooperate in a coordinated way. In the Artificial Intelligence literature, several research problems where proposed where these coordination strategies can be implemented and tested, e.g., robot soccer, rescue are surveillance activities, etc. One common artificial environment consists of a world in a grid format, that agents should explore to find resources, normally associated with "food". The idea of all these problems is to measure how the performance of the agents, in this case the quantity of food that is collected, increase with coordination, i.e., what is the impact of team work.

In this context, we have developed a software[1] that simulates a grid world, with twenty lines and twenty columns, where food can appear at a randomly chosen case of the world at regular time intervals. Four agents should coordinate their actions in the order to collect the maximimal quantity of food in a given period of time. Any agent that finds a food unit, should depose it into a given storage case. To evaluate the coordinate strategies we can modify the environment conditions to determine the better moment to use the strategy. The software was implemented in C++, using object orientation. The agents are implemented by a knowledge-based system, whose rules are explicitly codified into the software main program. As a base case, the performance without any cooperation strategy is considered. In this case, the agents search for food alone,

[1] The software can be downloaded from:
http://www.das.ufsc.br/~eder/clima.src.tar.gz.

without taking into account the other agents. As a first cooperation strategy we propose some actions to be taken when an agent finds food in the way to the depot.

2 The Environment: A Grid World

The environment consists of a grid, a matrix with twenty lines and twenty columns. The intersection between a line and a column is called a *case*. At each moment, each agent is located in exactly one case. One case is chosen to be the depot case, where all collected food should be stored. The simulation occurs in cycles, i.e., the temporal unit is a cycle. In a cycle, an agent can perform one, and only one, action, that can be either a movement to an adjacent case, or the emission of a message. Initially, the agents are located in the grid corners and the case depot is located in the center of the grid, in position (10,10).

According to the environment rules, one food unit appears automatically at every n cycles, where n is a simulation parameter. The experiments showed that the strategy efficacity varies with the parameter value. Others constraints were considered to create a problem more adequate to the multi-agent approach. All simulations were performed using a time interval of twenty seconds. In this way, we treat with a response pattern. The agent can carry only one food at a time. Once an agent collected a food, it can only leave it in the depot. There is no direct communication, all communication traffic is carried out by a communication manager. When an agent needs to send a message, this message is transmitted to a mailbox, and the communication manager delivers it to the message receiver. An agent can receive several messages, but it can participate in only one cooperation process at each time, i.e., during one cooperation process, the agent can not engage into a new cooperation process.

The software was implemented in the C++ programming language [1], using an object-oriented approach. A total of seven classes were implemented. The classes are: (i) **Position**: indicates a position in the grid; (ii) **Food**: contains the list of positions in the grid where the food is located; (iii) **Message**: contains constructors for messages with different performatives; (iv)**Mailbox**: used by the environment to manage the agent's messages; (v) **Clock**: implements the environment clock; (vi) **Interface**: verifies and validates the agent's actions in the environment, besides providing the input information for the agents; (vii)**Agent**: provides the agent internal variables and the actions in the environment, including movements and messages exchanges.

The main program, clima.cc, implements the four agents and their interactions with the environment are implemented, besides providing the interface with the exterior. This interface is in command-line form. In every cycle, the system updates the environment state, i.e, the simulation time, the cases with food, the agents positions, the number of foods collected, the number of food units already stored in the depot and the number of food units collected individually by each agent and the positions where they were collected.

For simplicity, the program was implemented in a totally sequential way. A more realistic approach would consider five different processes, one for for each agent and one for the environment, however the sequential approach response was satisfactory with the constraints described in this section.

3 The Cooperative Strategy

Each agent encapsulates a knowledge-based system. Nevertheless, the multi-agent system is a homogeneous one, where all the agents have the same knowledge about the domain. In each cycle, the agent is allowed to perform only one action, that can be either a movement to a new case, or a message emission to another agent, through the environment mailbox.

In a first strategy, there is no cooperation between the agents. Each agent search for food independently. The knowledges bases are build using the methodology described in [2], where a High-Level Petri net is used to describe the agent knowledge. This Petri net can be seen in the figure 1. In this first strategy, the transition $t5$ does not exist.[2]

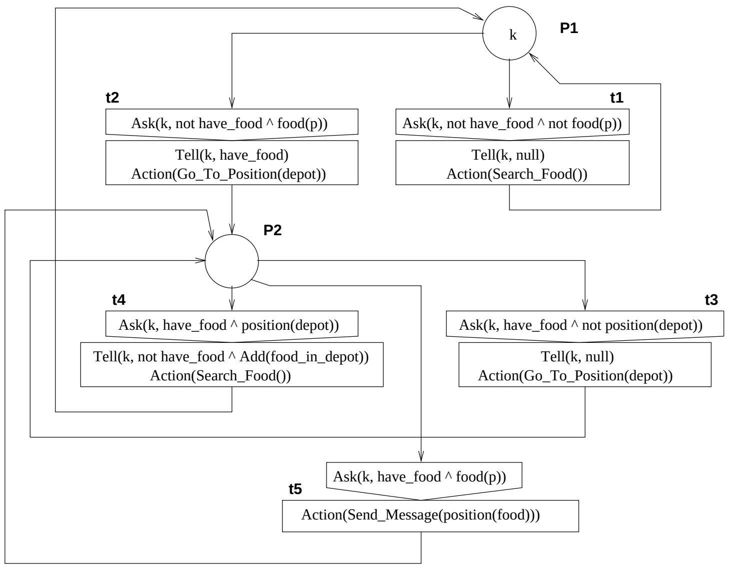

k
P1
t2
Ask(k, not have_food ^ food(p))
Tell(k, have_food)
Action(Go_To_Position(depot))
t1
Ask(k, not have_food ^ not food(p))
Tell(k, null)
Action(Search_Food())
P2
t4
Ask(k, have_food ^ position(depot))
Tell(k, not have_food ^ Add(food_in_depot))
Action(Search_Food())
t3
Ask(k, have_food ^ not position(depot))
Tell(k, null)
Action(Go_To_Position(depot))
Ask(k, have_food ^ food(p))
t5
Action(Send_Message(position(food)))

Fig. 1. The knowledge base of each agent represented by a Petri net.

The token represents the knowledge base (K). When this token enables a transition, an *Ask* is made to the K, that must return an answer. This answer

[2] The knowledge base represented in the Petri net is a simplified one. However, it is enough to understand the agent dynamic.

is represented by a directive $Tell$, where the inference engine returns the results to K. Besides that, an action is also inferred. This Petri net, in a next stage, is translated into a rule set that constitutes the knowledge base of each agent.

According to the knowledge base presented in figure 1, when an agent finds a food unit in its way to the depot case, it follows its way, and the found food unit is not collected. In a second strategy, when this state happens, a cooperation process is started, represented by the inclusion of the transition $t5$.

Now, when an agent finds a food unit in its way to the depot, it sends a message to the other agents with the food position. The agents answer this message telling the distance between them and the food position. The closest agent is considered the winner and starts a cooperation. When the food is stored in the depot, the cooperation is finished.

Following the FIPA-ACL [3], a message is constituted by a performative field, a sender field, a receiver field and a content field. The content field contains the agent position or the distance between the agent and the food. The sender and receiver fields contain, respectvilly, the identification of the agent that has sent the message and the identification of the agent that receives it. When the receiver field contain "all", all the agents receive the message.

The performative field describes the type of communication act intended with the message. A *request* is used when a cooperation is requested. A request should be answered with a *propose* or with a *refuse*. In the first case, the sender agent makes a proposal telling its distance from the food unit. In the second case, the agent is not ready to cooperate, because it is either carrying a food unit or involved in another cooperation process. The requesting agent receives the proposals and chooses the best one. In a last step, it sends a message with an *accept* performative to the winner and one with a *reject* performative to all the others. It is important to consider that only one message is delivered per cycle.

4 Results

The simulations were ran in a computer with Intel Celeron 2 Ghz processor, 256MB of RAM memory, running Mandrake Linux 10.0, using the gcc-2.96.

The performance of the system is measured by the number of cycles that the system needs to store a food. Considering a period of 80 cycles to appear a new food, the four agents, without any cooperation strategy needs 80.7 cycles to collect and store a food in the depot. If we consider the cooperation strategy the media is all the same: 80.6. The same results are obtained if we diminish the period down to 10 cycles. With a period of 40 cycles, the multi-agent system needs 41.1 cycles to collect and store one food unit, with or without the cooperation strategy.

With a period of 10 cycles between each new food unit, the strategy starts to make some difference. Without cooperation, the agents need 15.2 cycles to find and store a food unit. Using cooperation, this is reduced 14.8 cycles, still a minimum difference. When the period is set to 5 cycles, a greater difference appears: 10.9 cycles without cooperation and 10.1 cycles with cooperation. In fact, the smaller the cycle period, the greater is the number of times a conflict occurs. We consider a conflict when an agent carrying a food to the depot finds another food in its way.

5 Conclusion

There are many ways to cooperate in this problem. However this work looks to focus in one approach and analyzes its impact. This strategy consists in starting a cooperation strategy when an agent finds a food in its way to the depot. In this case, the agent is still carrying a resource, and is not able to take another one. In this case it sends a message to the the other agents to discover who is the closest agent. Once this agent is determined, it should go to food and collect it.

In fact, this is a special case inside the environment dynamics. It almost never happens. However, if the period between to succesive food introductions is diminished, the probability that this situation happens is increased, and the cooperation turns into an alternative to improve the system performance. This is the main conclusion of the results presented. The smaller the period between food appearances, the greater is the effect of the cooperation strategy.

It is important to see that it is not the only way to cooperate in this problem. Others ways include designate a fix collector that must collect the foods found by the others agents. In other case, when an agent find a food, it can ask to the others what they are doing. If there is an agent that is storing the food, it is informed with the position of the new food, and then goes to it. In others words, the collector role, in this case, is dynamic.

The approach presented here can model situations in real problems, like the collecting robots and intelligent routing in networks.

References

1. Stroustrup, B.: The C++ Programming Language. Addison Wesley Longman (1999) ISBN 0-201-88954-4.
2. Gonçalves, E.M.N., Bittencourt, G.: A planning-based knowledge acquisition methodology. In: Congress of Logic Applied to Technology (LAPTEC), IOS Press (2005)
3. Foundation for Intelligent Physical Agents http://www.fipa.org: FIPA ACL Message Structure Specification. (2002)

Author Index