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:

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

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

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:

$$L_1 ; \cdots ; L_l \leftarrow L_{l+1}, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n \quad (n \geq m \geq l \geq 0)$$

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* ${}^SP$ 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 ${}^SP$.

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 ${}^SP$ 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 ${}^SP \setminus P^S$, it holds $body^+(r) \subseteq S$, $(head(r) \leftarrow body^+(r)) \in P^S$ and $(head(r) \cap S \leftarrow body^+(r)) \in {}^SP$. As S satisfies P^S, $body^+(r) \subseteq S$ implies $head(r) \cap S \neq \emptyset$. So, S satisfies ${}^SP$. Assume that there is a minimal set $T \subset S$ satisfying every rule in ${}^SP$. Any rule r in $P^S \setminus {}^SP$ satisfies either (a) $body^+(r) \not\subseteq S$ or (b) $body^+(r) \subseteq S$, $(head(r) \leftarrow body^+(r)) \in P^S$ and $(head(r) \cap S \leftarrow body^+(r)) \in {}^SP$. 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 ${}^SP$, $body^+(r) \subseteq T$ 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 ${}^SP$. Hence, S is an answer set of P. The converse is shown in a similar manner. □

Example 2.1. Let P be the program:

$$p\,;\, q \leftarrow,$$
$$q \leftarrow p,$$
$$r \leftarrow not\, p.$$

For $S = \{q, r\}$, P^S becomes

$$p\,;\, q \leftarrow,$$
$$q \leftarrow p,$$
$$r \leftarrow,$$

while ${}^SP$ becomes

$$q \leftarrow,$$
$$r \leftarrow .$$

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

$$P_1 \,;\, \cdots \,;\, P_k =$$
$$\{\, head(r_1); \cdots ; head(r_k) \leftarrow body(r_1), \ldots, body(r_k) \;\mid\; r_i \in P_i \; (1 \leq i \leq k) \,\}.$$

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 $head(r) \cap body^{+}(r) \neq \emptyset$ (*elimination of tautologies*: TAUT); (ii) Delete a rule r from a program if there is another rule r' in the program such that $head(r') \subseteq head(r)$ and $body(r') \subseteq body(r)$ (*elimination of non-minimal rules*: NONMIN); (iii) A disjunction $(L; L)$ appearing in $head(r)$ is merged into L, and a conjunction (L, L) appearing in $body(r)$ 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:

$$\begin{aligned} P_1: \quad & p \leftarrow not\, q, \\ & q \leftarrow not\, p, \\ & s \leftarrow p, \\ P_2: \quad & p \leftarrow not\, r, \\ & r \leftarrow not\, p, \end{aligned}$$

where $\mathcal{AS}(P_1) = \{\{p, s\}, \{q\}\}$ and $\mathcal{AS}(P_2) = \{\{p\}, \{r\}\}$. 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, $(P_1 \odot P_2) \odot P_3 = P_1 \odot (P_2 \odot P_3)$. □

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. □

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 $(R(S,T)\,;\,R(S',T'))$. □

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, $U \in min(\mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2))$. □

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 $P_1 = \{\, p \leftarrow not\ \neg p \,\}$ and $P_2 = \{\, \neg p \leftarrow not\ p \,\}$. 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 $P_3 = \{\, p \leftarrow not\ q, \quad q \leftarrow not\ r \,\}$ and $P_4 = \{\, r \leftarrow not\ p \,\}$. 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, $P_1 \setminus {}^S P_1 = \{\, r \mid r \in P_1 \text{ and } body(r) \not\subseteq S \,\}$, 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, $\mathcal{AS}(P_1 \cup P_2) = \{Lit\}$. So, $S \cup T \subseteq Lit$. 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 $S \in \mathcal{AS}((P_1 \odot P_2) \cup IC(PB_1) \cup IC(PB_2))$. □

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 $P_1 = \{ \neg p \leftarrow not\, p \}$ 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 $q \leftarrow p$, $not\, q$ 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 con-

straints 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: $\{\, p \leftarrow \mathit{not}\, q,\quad q \leftarrow \mathit{not}\, p,\quad p \leftarrow \mathit{not}\, r,\quad r \leftarrow \mathit{not}\, p,\quad q \leftarrow p, \mathit{not}\, s,\quad s \leftarrow p, \mathit{not}\, q \,\}$. 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.