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.

Suppose a multi-agent system in which each agent has a knowledge base written in a common logic programming language. When two programs do not contradict each other, they are combined into one by taking the union of programs. The resulting program is the collection of knowledge of two agents, and extends the original program of each agent with additional information from the other one. In nonmonotonic logic programs, however, simple merging does not always reflect the meaning of individual programs. To see the problem, consider 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 either c_1 or c_2. On the other hand, another expert E_2 concludes that the trouble would be caused by either c_2 or c_3. E_1 (resp. E_2)

has no knowledge on the component c_3 (resp. c_1). Two experts' diagnoses are then encoded as the following logic programs:

$$E_1 : \quad c_1 \,;\, c_2 \leftarrow,$$
$$E_2 : \quad c_2 \,;\, c_3 \leftarrow,$$

where ; represents disjunction. By merging two programs, the program $E_1 \cup E_2$ has two *answer sets* $\{c_2\}$ and $\{c_1, c_3\}$. The first one is the common minimal model between two experts, while the second one is produced as a result of merging programs. Two solutions thus have different grounds and would be acceptable to each expert. The story goes on: E_1 considers that the possible cause is either c_1 or c_2, but he knows that c_1 is older and more likely to disorder. Similarly, E_2 resolves the possible cause into either c_2 or c_3, but she empirically knows that c_2 is fragile and more likely to cause the trouble. Two experts then modify their diagnoses as

$$E_1' : \quad c_1 \leftarrow not\, c_2,$$
$$c_2 \leftarrow \neg\, c_1,$$
$$E_2' : \quad c_2 \leftarrow not\, c_3,$$
$$c_3 \leftarrow \neg\, c_2,$$

where *not* represents *negation as failure*. 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 proved to be false. E_2' is read in a similar way. Merging two programs, however, $E_1' \cup E_2'$ 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 consensus of two experts. The problem is explained as follows: c_1 in E_1' and c_2 in E_2' are both *default* consequences derived from incomplete information in each program. However, simple merging has the effect of *preferring* c_2 to c_1 as the former is included in a relatively lower stratum than the latter. In logic programming consequences derived from a lower stratum are preferred in a single program, but the principle is not necessarily applied to the case of combining different programs. As observed in the above example, the local preference in E_1' or E_2' does not necessarily imply the global preference in $E_1' \cup E_2'$.

Thus, 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* [16]. 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 [3]. A program may have multiple answer sets, and different agents have different collections of answer sets in general. 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 $\mathcal{AS}(Q) = min(\mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2))$ 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 $min(X) = \{\, Y \in X \mid \neg\exists Z \in X \text{ s.t. } Z \subset Y \,\}$. The program Q satisfying the above condition is called a *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. Finally, we apply the theory to a logical formulation of multi-agent coordination.

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* [16]. Let *Lit* be the set of all ground literals in the language of a program. A set $S(\subseteq Lit)$ *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 a ground NAF-free EDP. Then, a set $S(\subseteq Lit)$ is an *answer set* of P if S is a minimal set such that (i) S satisfies every rule from P; and (ii) if S contains a pair of complementary literals L and $\neg L$, $S = Lit$. Next, let P be any ground EDP and $S \subseteq Lit$. For every rule r in 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 different from the original one in [16]. In [16], the rule $head(r) \leftarrow body^+(r)$ is included in the reduct P^S (called Gelfond-Lifschitz reduction) if $body^-(r) \cap S = \emptyset$. A similar but different definition of reduct is in [15], where the rule $head(r) \leftarrow body^+(r)$ is included in the reduct if $body^+(r) \subseteq S$ and $body^-(r) \cap S = \emptyset$. Thus, disjunctive heads remain unchanged in the definition of [15].

Our reduction imposes additional conditions, but it produces the same answer sets as Gelfond-Lifschitz reduction does.

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 $body(r) = body^+(r) \subseteq S$ and $(head(r)' \leftarrow body(r)) \in P^S$ with $head(r) = head(r)' \cap S$. 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 ${}^S P$.

Conversely, if S is an answer set of ${}^S P$, S is a minimal set satisfying every rule in ${}^S P$. For any rule r in $P^S \setminus {}^S P$, it holds either (a) $body(r) \not\subseteq S$ or (b) $body(r) \subseteq S$, $(head(r) \leftarrow body(r)) \in P^S$ and $(head(r) \cap S \leftarrow body(r)) \in {}^S P$. As S satisfies ${}^S P$, $body(r) \subseteq S$ implies $head(r) \cap S \neq \emptyset$. Thus, in each case S satisfies every rule r in P^S. Assume that there is a minimal set $T \subset S$ satisfying every rule in P^S. For any rule r in ${}^S P \setminus P^S$, it holds $body(r) \subseteq S$ and $(head(r)' \leftarrow body(r)) \in P^S$ with $head(r) = head(r)' \cap S$. If $body(r) \not\subseteq T$, T satisfies r. Else if $body(r) \subseteq T$, $head(r)' \cap T \neq \emptyset$. As $head(r)' \cap T = head(r)' \cap S \cap T$, it holds that $head(r)' \cap S \cap T = head(r) \cap T \neq \emptyset$. Hence, T satisfies $head(r) \leftarrow body(r)$ in ${}^S P$. This contradicts the fact that S is a minimal set satisfying ${}^S P$. Then, S is an answer set of ${}^S P$. $\square$

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.

The new reduct ^SP has the effect of (i) reducing any rule in P that is irrelevant to constructing S, and (ii) eliminating any disjunct in the head of a rule that is not a consequence in S. For technical reasons, we use the reduct ^SP for computing answer sets of P.[3]

A program has none, one, or multiple answer sets in general. The set of all answer sets of P is written as $\mathcal{AS}(P)$. Every element in $\mathcal{AS}(P)$ is *minimal*, that is, $S \subseteq T$ implies $T \subseteq S$ for any S and T in $\mathcal{AS}(P)$. A program having a single answer set is called *categorical* [3]. Categorical programs include important classes of programs such as *definite programs*, *stratified programs*, and *call-consistent programs*. Every NAF-free ELP has at most one 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*. An inconsistent program has either no answer set or the single answer set *Lit*.

Proposition 2.2 *If a program P is consistent, ^SP contains no integrity constraint for any $S \in \mathcal{AS}(P)$.*

Proof. By the definition, for any integrity constraint $r \in P$, $\leftarrow body^+(r)$ is included in ^SP if $body^+(r) \subseteq S$ and $body^-(r) \cap S = \emptyset$. In this case, however, S does not satisfy r, so that it is not an answer set of P. Contradiction. $\square$

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 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 consistent program P.

Example 2.2. Let P be the program:

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

where $\mathcal{AS}(P) = \{\{p, r\}, \{q, r\}\}$. Then, $crd(P) = \{p, q, r\}$ and $skp(P) = \{r\}$.

---

[3] We will address the effect of this new reduct in Section 4.

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.

Definition 3.1. 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

$$\mathcal{S} \uplus \mathcal{T} = \{ S \uplus T \mid S \in \mathcal{S} \text{ and } T \in \mathcal{T} \}.$$

In particular, $\mathcal{S} \uplus \mathcal{T} = \emptyset$ if $\mathcal{S} = \emptyset$ or $\mathcal{T} = \emptyset$.

Definition 3.2. Let P_1 and P_2 be two 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. Note that the operation $min(\cdot)$ has the effect of making every element in $\mathcal{AS}(Q)$ incomparable under set inclusion.

Example 3.1. Let $\mathcal{AS}(P_1) = \{\{p\}, \{q\}\}$ and $\mathcal{AS}(P_2) = \{\{p\}, \{r\}\}$. Then, the compositional semantics becomes $\mathcal{AS}(Q) = \{ \{p\}, \{q, r\} \}$.

In what follows, when we refer a program Q to a composition of P_1 and P_2, it means a program Q satisfying the condition of Definition 3.2.

When categorical programs are composed, the resulting program is also categorical.

Proposition 3.1 *If P_1 and P_2 are two categorical programs, a composition Q of P_1 and P_2 is categorical.*

Proof. Let $\mathcal{AS}(P_1) = \{S\}$ and $\mathcal{AS}(P_2) = \{T\}$. Then, the compositional semantics becomes $\mathcal{AS}(Q) = \{ S \cup T \}$ if $S \cup T$ is consistent; otherwise, $\mathcal{AS}(Q) = \{ Lit \}$. In each case, Q has the single answer set, thereby categorical.   □

The following properties directly hold by the definition.

Proposition 3.2 *Let P_1 and P_2 be programs, and Q a composition of P_1 and P_2. Then,*

1. $\mathcal{AS}(P_1) = \{Lit\}$ and $\mathcal{AS}(P_2) \neq \emptyset$ imply $\mathcal{AS}(Q) = \{Lit\}$.
2. $\mathcal{AS}(P_1) = \emptyset$ or $\mathcal{AS}(P_2) = \emptyset$ implies $\mathcal{AS}(Q) = \emptyset$.

As shown in Proposition 3.2, if one of two programs is inconsistent, the result of composition is rather trivial. We thus consider compositions of consistent programs hereafter.

Proposition 3.3 *Let P_1 and P_2 be two consistent programs, and Q a composition of P_1 and P_2. Then, for any $S \in \mathcal{AS}(Q)$, there is $T \in \mathcal{AS}(P_i)$ for $i = 1, 2$ such that $T \subseteq S$.*

Proof. If Q is consistent, for any $S \in \mathcal{AS}(Q)$ there exists $T \in \mathcal{AS}(P_1)$ and $T' \in \mathcal{AS}(P_2)$ such that $S = T \cup T'$ and $T \cup T'$ is consistent. Then, $T \subseteq S$ and $T' \subseteq S$ hold. Else if Q is inconsistent, $\mathcal{AS}(Q) = \{Lit\}$. Then, $T \subset Lit$ and $T' \subset Lit$ for any $T \in \mathcal{AS}(P_1)$ and any $T' \in \mathcal{AS}(P_2)$. $\square$

Proposition 3.3 asserts 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 3.3. Let P_1 and P_2 be two consistent programs, and Q a composition of P_1 and P_2. 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 3.4 *Let P_1 and P_2 be two consistent programs, and Q a composition of P_1 and P_2. 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 $T' \nsubseteq S$ 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, $\mathcal{AS}(Q) \backslash \mathcal{AS}(P_1) = \emptyset$. 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, $T \subseteq S$. $\square$

Skeptical/credulous inference in compositional semantics has the following properties.

Proposition 3.5 *Let P_1 and P_2 be two consistent programs, and Q a composition of P_1 and P_2. When Q is consistent, the following relations hold.*

1. $crd(Q) \subseteq crd(P_1) \cup crd(P_2)$.
2. $skp(Q) = skp(P_1) \cup skp(P_2)$.

Proof. (1) Any literal included in a consistent answer set $S \in \mathcal{AS}(Q)$ is either included in an answer set $T \in \mathcal{AS}(P_1)$ or included in an answer set $T' \in \mathcal{AS}(P_2)$. (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 consistent 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 $L \notin S'$. If there is $T' \in \mathcal{AS}(P_2)$ such that $L \notin T'$, then $L \notin S' \cup T'$ so there is $V \in \mathcal{AS}(Q)$ such that $L \notin V \subseteq S' \cup T'$. Contradiction. Hence, $L \in T$ for every $T \in \mathcal{AS}(P_2)$. □

Thus, if the compositional semantics is consistent, it combines skeptical consequences of P_1 and P_2, and any information included in an answer set of Q has its origin in an answer set of P_1 or P_2. The above relations do not hold when Q is inconsistent.

Example 3.3. Let $\mathcal{AS}(P_1) = \{\{p, a\}, \{p, b\}\}$ and $\mathcal{AS}(P_2) = \{\{\neg p, a\}, \{\neg p, b\}\}$ where $crd(P_1) = \{p, a, b\}$, $skp(P_1) = \{p\}$, $crd(P_2) = \{\neg p, a, b\}$, and $skp(P_2) = \{\neg p\}$. The compositional semantics of P_1 and P_2 becomes $\mathcal{AS}(Q) = \{Lit\}$ where $crd(Q) = skp(Q) = Lit$.

As observed in the above example, the result of composition may become inconsistent even if the original programs are consistent. 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 3.6 *Let P_1 and P_2 be consistent programs, and Q a composition of P_1 and P_2. 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 case 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.

Definition 4.1. Let $P_1, \ldots, P_k$ be programs. Then, define

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

Thus, $P_1 \,;\, \cdots \,;\, P_k$ is an EDP which is obtained by disjunctively combining any rule from P_i $(1 \le i \le k)$ in every possible way. When all programs are NAF-free, the following properties hold.

Proposition 4.1 *Let $P_1, \ldots, P_k$ be NAF-free programs. Then, $\mathcal{AS}(P_1 \,;\, \cdots \,;\, P_k) = min(\mathcal{AS}(P_1) \cup \cdots \cup \mathcal{AS}(P_k))$.*

Proof. Let $S \in min(\mathcal{AS}(P_1) \cup \cdots \cup \mathcal{AS}(P_k))$. Then, $S \in \mathcal{AS}(P_i)$ for some $1 \le i \le k$. So, for any $r_i \in P_i$, either $head(r_i) \cap S \ne \emptyset$ or $body(r_i) \not\subseteq S$ holds. Correspondingly, for any $R = head(r_1); \cdots; head(r_k) \leftarrow body(r_1), \ldots, body(r_k)$ in $P_1 \,;\, \cdots \,;\, P_k$, either $head(R) \cap S \ne \emptyset$ or $body(R) \not\subseteq S$ holds. Hence, S satisfies every rule in $P_1 \,;\, \cdots \,;\, P_k$. Next, suppose that there is a minimal set $T \subset S$ which satisfies every rule in $P_1 \,;\, \cdots \,;\, P_k$. Then, for any rule R of the above form, either $head(R) \cap T \ne \emptyset$ or $body(R) \not\subseteq T$ holds. On the other hand, by $T \notin min(\mathcal{AS}(P_1) \cup \cdots \cup \mathcal{AS}(P_k))$, T satisfies no P_i. Then, for any P_i, there is a rule $r_i \in P_i$ such that $head(r_i) \cap T = \emptyset$ and $body(r_i) \subseteq T$. However, every such rule is combined into a rule $R = head(r_1); \cdots; head(r_k) \leftarrow body(r_1), \ldots, body(r_k)$ in $P_1 \,;\, \cdots \,;\, P_k$, and it holds that $head(R) \cap T = \emptyset$ and $body(R) \subseteq T$. Contradiction. Hence, S is a minimal set satisfying every rule in $P_1 \,;\, \cdots \,;\, P_k$, and $S \in \mathcal{AS}(P_1 \,;\, \cdots \,;\, P_k)$.

Conversely, let $S \in \mathcal{AS}(P_1 \,;\, \cdots \,;\, P_k)$. If S satisfies no P_i $(1 \le i \le k)$, every P_i contains a rule r_i such that $head(r_i) \cap S = \emptyset$ and $body(r_i) \subseteq S$. Every such rule is combined into $R = head(r_1); \cdots; head(r_k) \leftarrow body(r_1), \ldots, body(r_k)$ in $P_1 \,;\, \cdots \,;\, P_k$, and it holds that $head(R) \cap S = \emptyset$ and $body(R) \subseteq S$. So S does not satisfy $P_1 \,;\, \cdots \,;\, P_k$. Contradiction. Hence, S satisfies some P_i $(1 \le i \le k)$. Next, suppose that there is a minimal set $T \subset S$ satisfying P_i. Then, for any $r_i \in P_i$, it holds that either $head(r_i) \cap T \ne \emptyset$ or $body(r_i) \not\subseteq T$. In this case, T satisfies every $head(r_1); \cdots; head(r_k) \leftarrow body(r_1), \ldots, body(r_k)$ in $P_1 \,;\, \cdots \,;\, P_k$. This contradicts the fact that S is a minimal set satisfying $P_1 \,;\, \cdots \,;\, P_k$. Hence, $S \in \mathcal{AS}(P_i)$. Suppose that $S \notin min(\mathcal{AS}(P_1) \cup \cdots \cup \mathcal{AS}(P_k))$. Then, there is $S' \in \mathcal{AS}(P_j)$ $(1 \le j \le k)$ such that $S' \subset S$ and $S' \in min(\mathcal{AS}(P_1) \cup \cdots \cup \mathcal{AS}(P_k))$. In this case, $S' \in \mathcal{AS}(P_1 \,;\, \cdots \,;\, P_k)$ by the above proof. But this cannot happen, since $S \in \mathcal{AS}(P_1 \,;\, \cdots \,;\, P_k)$. Hence, $S \in min(\mathcal{AS}(P_1) \cup \cdots \cup \mathcal{AS}(P_k))$. $\qquad\square$

Corollary 4.2 *Let $P_1, \ldots, P_k$ be NAF-free programs. Then, $P_1 \,;\, \cdots \,;\, P_k$ is consistent iff some P_i $(1 \le i \le k)$ is consistent.*

Proof. The result follows from Proposition 4.1. $\qquad\square$

Definition 4.2. Let P_1 and P_2 be two programs such that $\mathcal{AS}(P_1) = \{\, S_1, \ldots, S_m \,\}$ 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 $R(S,T) = {}^{S}P_1 \cup {}^{T}P_2$ and $R(S_1,T_1),\ldots,R(S_m,T_n)$ is any enumeration of the $R(S_i,T_j)$'s for $S_i \in \mathcal{AS}(P_1)$ $(i=1,\ldots,m)$ and $T_j \in \mathcal{AS}(P_2)$ $(j=1,\ldots,n)$. In particular, $R(S,T) = \emptyset$ when $\mathcal{AS}(P_i) = \emptyset$ for $i=1$ or $i=2$.

$R(S,T)$ merges every NAF-free rule which contributes to the construction of an answer set S of P_1 and T of P_2. Those rules are then disjunctively combined for any $S_i \in \mathcal{AS}(P_1)$ and for any $T_j \in \mathcal{AS}(P_2)$ in every possible way. By the definition, $P_1 \odot P_2$ is computed in time $O(|P_1| \times |P_2| \times |\mathcal{AS}(P_1)| \times |\mathcal{AS}(P_2)|)$, 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 operator $\odot$ has the following properties.

Proposition 4.3 *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, $(P_1 \odot P_2) \odot P_3 = P_1 \odot (P_2 \odot P_3)$. $\square$

Proposition 4.4 *Let P_1 and P_2 be programs. Then,*

1. *$\mathcal{AS}(P_1) = \{Lit\}$ and $\mathcal{AS}(P_2) \neq \emptyset$ imply $\mathcal{AS}(P_1 \odot P_2) = \{Lit\}$.*
2. *$\mathcal{AS}(P_1) = \emptyset$ or $\mathcal{AS}(P_2) = \emptyset$ implies $\mathcal{AS}(P_1 \odot P_2) = \emptyset$.*

Proof. (1) When $\mathcal{AS}(P_1) = \{Lit\}$, it becomes $P_1 \odot P_2 = R(Lit,T_1)\,;\,\cdots\,;\,R(Lit,T_n)$ for $\mathcal{AS}(P_2) = \{T_1,\ldots,T_n\}$. Here, every $R(Lit,T_i)$ $(1 \leq i \leq k)$ has the answer set Lit, so that the result follows by Proposition 4.1. (2) When $\mathcal{AS}(P_1) = \emptyset$, $R(S,T) = \emptyset$ for any $T \in \mathcal{AS}(P_2)$ by Definition 4.2. Then, $\mathcal{AS}(P_1 \odot P_2) = \emptyset$ by Proposition 4.1. $\square$

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 [5].

Example 4.1. Consider two programs:

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

where $\mathcal{AS}(P_1) = \{\{p, s\}, \{q\}\}$ and $\mathcal{AS}(P_2) = \{\{p\}, \{r\}\}$. There are four $R(S, T)$'s such that

$$
\begin{aligned}
R(\{p, s\}, \{p\}) : \quad & p \leftarrow, \quad s \leftarrow p, \\
R(\{p, s\}, \{r\}) : \quad & p \leftarrow, \quad s \leftarrow p, \quad r \leftarrow, \\
R(\{q\}, \{p\}) : \quad & q \leftarrow, \quad p \leftarrow, \\
R(\{q\}, \{r\}) : \quad & q \leftarrow, \quad r \leftarrow .
\end{aligned}
$$

Then, $P_1 \odot P_2$ contains the following seven rules (after applying DUPL):

$$
\begin{aligned}
& p \, ; \, q \leftarrow, && (1) \\
& p \, ; \, r \leftarrow, && (2) \\
& p \, ; \, q \, ; \, r \leftarrow, && (3) \\
& q \, ; \, s \leftarrow p, && (4) \\
& q \, ; \, r \, ; \, s \leftarrow p, && (5) \\
& p \, ; \, q \, ; \, s \leftarrow p, && (6) \\
& p \, ; \, r \, ; \, s \leftarrow p. && (7)
\end{aligned}
$$

Further, rules (3), (5), (6), and (7) are eliminated by NONMIN. Consequently, the simplified program becomes

$$
\begin{aligned}
& p \, ; \, q \leftarrow, \\
& p \, ; \, r \leftarrow, \\
& q \, ; \, s \leftarrow p.
\end{aligned}
$$

In the resulting program, the first rule $p \, ; \, q \leftarrow$ corresponds to the rules $p \leftarrow not\, q$ and $q \leftarrow not\, p$ in P_1. The second rule $p \, ; \, r \leftarrow$ corresponds to the rules $p \leftarrow not\, r$ and $r \leftarrow not\, p$ in P_2. On the other hand, one might wonder the effect of q in the head of the third rule $q \, ; \, s \leftarrow p$. Without q, however, the set $\{p, q\}$, which is obtained by combining $\{q\} \in \mathcal{AS}(P_1)$ and $\{p\} \in \mathcal{AS}(P_2)$, does not become an answer set of the resulting program.

Now we show that the operator $\odot$ computes a composition of P_1 and P_2.

Lemma 4.5 *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, $S \uplus T$ is an answer set of $^S P_1 \cup {}^T P_2$.*

Proof. As P_1 and P_2 is consistent, neither $^S P_1$ nor $^T P_2$ contains integrity constraints (Proposition 2.2). When the NAF-free program $^S P_1 \cup {}^T P_2$ is inconsistent, it has the answer set Lit. Suppose that $S \cup T$ is consistent. Since S satisfies every rule in $^S P_1$ and T satisfies every rule in $^T P_2$, $S \cup T$ satisfies $^S P_1 \cup {}^T P_2$. Contradiction. So $S \cup T$ is inconsistent. Then, $S \uplus T = S \cup T = Lit$, and the result holds. Next, consider the case that $^S P_1 \cup {}^T P_2$ is consistent. Then, S is a minimal set satisfying $^S P_1$ and T is a minimal set satisfying $^T P_2$. As (i) $body(r) \subseteq S$ and $head(r) \subseteq S$ for any $r \in {}^S P_1$, and (ii) $body(r') \subseteq T$ and $head(r') \subseteq T$ for any $r' \in {}^T P_2$, it holds that $S \cup T$ 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 and consistent, $S \uplus T = S \cup T$ becomes an answer set of it. $\qquad\square$

It is worth noting that the above lemma does not hold if we use Gelfond-Lifschitz reduction P^S instead of ${}^S P$. This is because $P_1^S \cup P_2^T$ may derive literals which are not in $S \cup T$. This is the reason why we use a new reduct in this paper.

Theorem 4.6. *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 $U \in min(\mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2))$. (i) If $U = Lit$, $S \cup T$ is inconsistent for any $S \in \mathcal{AS}(P_1)$ and for any $T \in \mathcal{AS}(P_2)$ (Proposition 3.6). Then, $R(S,T)$ has the answer set Lit for any $S \in \mathcal{AS}(P_1)$ and for any $T \in \mathcal{AS}(P_2)$ (Lemma 4.5), so $\mathcal{AS}(P_1 \odot P_2) = \{Lit\}$ by Proposition 4.1. (ii) Else if $U \neq Lit$, there is $S \in \mathcal{AS}(P_1)$ and $T \in \mathcal{AS}(P_2)$ such that $U = S \cup T$ is consistent (Proposition 3.6). By Lemma 4.5, 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 $P_1 \odot P_2$ (Proposition 4.1). 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 4.5). 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 $U \in \mathcal{AS}(P_1 \odot P_2)$. (i) If $U = Lit$, $R(S,T)$ is inconsistent for any $S \in \mathcal{AS}(P_1)$ and for any $T \in \mathcal{AS}(P_2)$ (by Corollary 4.2). Then, $S \cup T$ is inconsistent for any $S \in \mathcal{AS}(P_1)$ and for any $T \in \mathcal{AS}(P_2)$ (Lemma 4.5), thereby $min(\mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2)) = \{Lit\}$. (ii) Else if $U \neq Lit$, U is a consistent minimal set satisfying some $R(S,T)$ in $P_1 \odot P_2$ (Proposition 4.1). It then holds $U = S \cup T$ for some $S \in \mathcal{AS}(P_1)$ and $T \in \mathcal{AS}(P_2)$ (by Lemma 4.5). 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, $U \in min(\mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2))$. $\qquad\square$

Corollary 4.7 *Let P_1 and P_2 be two categorical programs such that $\mathcal{AS}(P_1) = \{S\}$ and $\mathcal{AS}(P_2) = \{T\}$. Then, $\mathcal{AS}(P_1 \odot P_2) = \{S \cup T\}$.*

Proof. When P_1 and P_2 are consistent, the result follows by Theorem 4.6. Suppose that either P_1 or P_2 is inconsistent. Let $\mathcal{AS}(P_1) = \{Lit\}$. Then, ${}^{Lit} P_1$ is inconsistent. By $P_1 \odot P_2 = {}^{Lit} P_1 \cup {}^T P_2$, $P_1 \odot P_2$ is also inconsistent. Moreover, ${}^{Lit} P_1 \cup {}^T P_2$ contains no integrity constraint (Proposition 2.2), so that it has the inconsistent answer set Lit. Hence, the result holds. $\qquad\square$

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 4.8 *For two 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, $P_1 \odot P_2$ is consistent by Corollary 4.2. □

The converse of Proposition 4.8 does not hold in general.

Example 4.3. Let $P_1 = \{ p \leftarrow \}$ and $P_2 = \{ \neg p \leftarrow p \}$. Then, $P_1 \odot P_2 = \{ p \leftarrow \}$, but $P_1 \cup P_2$ has the answer set *Lit*.

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 $P_1 \odot P_2 = \{ p \leftarrow \, , \quad \neg p \leftarrow \}$ 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 4.9 *For two NAF-free ELPs P_1 and P_2, $P_1 \odot P_2 \subseteq P_1 \cup P_2$.*

Proof. Any NAF-free ELP has at most one answer set. If $\mathcal{AS}(P_1) \neq \emptyset$ and $\mathcal{AS}(P_2) \neq \emptyset$, let $\mathcal{AS}(P_1) = \{S\}$ and $\mathcal{AS}(P_2) = \{T\}$. Then, $P_1 \setminus {}^S P_1 = \{ r \mid r \in P_1 \text{ and } body(r) \nsubseteq S \}$, 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. Else if $\mathcal{AS}(P_1) = \emptyset$ or $\mathcal{AS}(P_2) = \emptyset$, $P_1 \odot P_2 \subseteq {}^S P_1 \cup {}^T P_2$. Then, the result also holds. □

Proposition 4.10 *Let P_1 and P_2 be two consistent NAF-free ELPs. If $\mathcal{AS}(P_1 \cup P_2) \neq \emptyset$, 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, $\mathcal{AS}(P_1 \odot P_2) = \{S \cup T\}$. 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 4.8. Then, $S \cup T \subset V$ by Proposition 4.9. □

Example 4.5. Let $P_1 = \{ p \leftarrow q \}$ and $P_2 = \{ q \leftarrow \}$. Then, $P_1 \odot P_2 = \{ q \leftarrow \}$ and $P_1 \cup P_2 = \{ p \leftarrow q, \quad q \leftarrow \}$. 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 composition of P_1 and P_2. Then, any answer set $S \in \mathcal{AS}(Q)$ is *consenting* if it satisfies every rule in $P_1 \cup P_2$.

Example 5.1. Recall two programs P_1 and P_2 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 of P_1 and P_2 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 consenting. Note that $\{p, q\}$ is not consenting because it does not satisfy the third rule of P_1.

Consenting answer sets are good candidates for coordinative solutions, because they satisfy the original program of each agent. A consenting answer set is possibly inconsistent. Unfortunately, consenting answer sets do not always exist in the compositional semantics. For instance, in Example 5.1 if P_2 contains integrity constraints $\leftarrow s$ and $\leftarrow q$, no consenting answer set exists. Existence of no consenting answer set in general is not a serious flaw in the compositional semantics, however. In fact, different agents have different beliefs in a multi-agent environment, and it may happen that one agent must give up some original belief to reach an acceptable 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, persistent 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 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 (sustaining PB_1 and PB_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 3.2. The permissible compositional semantics reduces to the compositional semantics when $PB_1 \cup PB_2 = \emptyset$. In particular, consenting 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 some answer sets of an agent by additional information of another agent.

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 $IC(P) = \{ \leftarrow body(r), not\_head(r) \mid r \in P \}$ where $not\_head(r)$ is the conjunction of NAF-literals $\{ not\, L_1, \ldots, not\, L_l \}$ for $head(r) = \{ L_1, \ldots, L_l \}$.

Theorem 5.1. *Let P_1 and P_2 be consistent programs, and Ω a permissible composition of P_1 and P_2. 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 4.6, $S \in \mathcal{AS}(\Omega)$
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 $S \in \mathcal{AS}((P_1 \odot P_2) \cup IC(PB_1) \cup IC(PB_2))$. $\qquad\square$

Example 5.2. Consider two programs P_1 and P_2 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

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

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 [7, 12] 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 [7]. 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" [7]. 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 [6, 8, 9, 14, 23].

Compared with those previous studies, our approach is different in the following aspects. First, our primary interest is not simply merging two programs but building a new program that combines answer sets of the original programs. Second, our program composition is intended to coordinate meanings of different programs, rather than to synthesize a program by its component. 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 a composed program as the answer set $\{p\}$ of $P_1 \cup P_2$. By contrast, our compositional semantics $P_1 \odot P_2$ becomes inconsistent, that is, 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 two programs. 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. For another example, consider $P_3 = \{ p \leftarrow q \}$ and $P_4 = \{ p \leftarrow not\, q, \quad q \leftarrow \}$. Two agents have incompatible beliefs; P_3 believes that p holds if q holds, while P_4 believes that p holds if q does not hold. Now P_4 knows q, so that p is not believed. Merging two programs, however, p is derived from $P_3 \cup P_4$. This is rather an unexpected consequence for P_4. As argued in the introduction, simple merging of different programs does not always reflect the meaning of individual programs. We then took an approach of retaining belief of each agent and combine answer sets of different programs. As a result, the compositional semantics maintains information included in (at least one) answer set of the original programs. It precisely combines the results of skeptical consequences of original programs and does not introduce additional (unexpected) consequences (Proposition 3.5). Note that program composition should be distinguished from *revision* or *update*, in which one of two information sources 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. Several studies argue combining different theories having priorities [2, 11, 13, 19, 21]. Priorities are useful to resolve conflicts among agents, however, it generally

introduces additional computational cost. Our compositional semantics does not handle programs with different priorities, but prioritized coordination is partly realized by permissible composition. If P_1 is more reliable than P_2, P_1 is put as persistent beliefs. Under the setting, every permissible answer set satisfies 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. Moreover, 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 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* [18] or *arbitration* [20], but they do not handle nonmonotonic theories.

Buccafurri and Gottlob [10] introduce a framework of *compromise logic programs* which aims at reaching common conclusions and compromises among logic programming agents. Given a collection of programs $T = \{Q_1, \ldots, Q_n\}$, the joint fixpoints $JFP(T)$ is defined as $JFP(T) = FP(Q_1) \cap \cdots \cap FP(Q_n)$ where $FP(Q_i)$ is the set of all fixpoints of Q_i. Then, the *joint fixpoint semantics* of T is defined as the set of minimal elements in $JFP(T)$. The joint fixpoint semantics is different from our compositional semantics. For instance, when two programs $P_1 = \{ p \leftarrow \}$ and $P_2 = \emptyset$ are given, by $FP(P_1) = \{\{p\}\}$ and $FP(P_2) = \{\emptyset\}$ their joint fixpoint semantics becomes $\emptyset$. Interestingly, however, if a tautology $p \leftarrow p$ is added to P_2, $FP(P_2)$ turns to $\{\emptyset, \{p\}\}$ and the joint fixpoint semantics becomes $\{\{p\}\}$. Thus, in their framework a rule $p \leftarrow p$ has a special meaning that "if p is required by another agent, let it be". With this reading, however, $P_1 = \{ p \leftarrow \}$ and $P_3 = \{ p \leftarrow p, \quad q \leftarrow \}$ have the joint fixpoint semantics $\{\emptyset\}$, that is, P_3 does not tolerate p when another irrelevant fact q exists in the program. By contrast, our compositional semantics becomes $\mathcal{AS}(P_1 \odot P_2) = \{\{p\}\}$ and $\mathcal{AS}(P_1 \odot P_3) = \{\{p, q\}\}$.

Sakama and Inoue [22] 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. Aside from such differences, the present work is also applied to coordinate agents, so that 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*, that is, it contains no positive cycle through disjuncts appearing in the head of a disjunctive rule [4]. 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 but leaving one in the head. For instance, the program $P_1 \odot P_2$ in Example 4.1 is converted to the ELP: $\{\, p \leftarrow not\, q, \quad q \leftarrow not\, p, \quad p \leftarrow not\, r, \quad r \leftarrow not\, p, \quad q \leftarrow p, not\, s, \quad s \leftarrow p, not\, q \,\}$. The resulting program has the same answer sets as the original disjunctive program.

7 Conclusion

This paper has studied a 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, however. This paper considered compositional semantics as minimal sets that reflect the meaning of original programs. By contrast, a program may have *non-minimal* answer sets in the context of *general extended disjunctive programs* which possibly contain NAF in the heads of rules [17]. In this context, the compositional semantics would be defined as a collection of non-minimal answer sets. These extensions and variants of compositional semantics will be investigated in future study.

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, S. Kraus, J. Minker, and V. S. Subrahmanian. Combining knowledge base consisting of first-order theories. *Computational Intelligence*, 8:45–71, 1992.
3. C. Baral and M. Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19/20:73–148, 1994.
4. R. Ben-Eliyahu and R. Dechter. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 12(1):53–87, 1994.
5. S. Brass and J. Dix. Characterizations of the disjunctive stable semantics by partial evaluation. *Journal of Logic Programming*, 32(3):207–228, 1997.
6. A. Brogi, S. Contiero, and F. Turini. Programming by combining general logic programs. *Journal of Logic and Computation*, 9(1):7–24, 1999.
7. 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.
8. 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.
9. 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.
10. F. Buccafurri and G. Gottlob. Multiagent compromises, joint fixpoints, and stable models. *Computational Logic: Logic Programming and Beyond*, Lecture Notes in Artificial Intelligence 2407, pp. 561–585, Springer, 2002.
11. F. Buccafurri, W. Faber, and N. Leone. Disjunctive programs with inheritance. *Proceedings of the 1999 International Conference on Logic Programming*, pp. 79–93, MIT Press, 1999.
12. M. Bugliesi, E. Lamma, and P. Mello. Modularity in logic programming. *Journal of Logic Programming*, 19/20:443–502, 1994.
13. M. De. Vos and D. Vermeir. Extending answer sets for logic programming agents. *Annals of Mathematics and Artificial Intelligence*, 42: 103–139, 2004.
14. 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.
15. W. Faber, N. Leone, and G. Pfeifer. Recursive aggregates in disjunctive logic programs: semantics and complexity. *Proceedings of the 9th European Conference on Logics in Artificial Intelligence*, Lecture Notes in Artificial Intelligence, 3229, pp. 200–212, Springer, 2004.
16. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–385, 1991.
17. K. Inoue and C. Sakama. Negation as failure in the head. *Journal of Logic Programming*, 35(1):39–78, 1998.
18. 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.
19. J. A. Leite. *Evolving Knowledge Bases, Specification and Semantics*. IOS Press, 2003.

20. P. Liberatore and M. Schaerf. Arbitration (or how to merge knowledge bases). *IEEE Transactions on Knowledge and Data Engineering*, 10(1):76–90, 1998.

21. S. Pradhan and J. Minker. Using priorities to combine knowledge bases. *Journal of Cooperative Information Systems*, 5(2&3):333–364. 1996.

22. C. Sakama and K. Inoue. Coordination between logical agents. *Proceedings of the 5th International Workshop on Computational Logic in Multi-Agent Systems*, Lecture Notes in Artificial Intelligence 3487, pp. 161–177, Springer, 2005.

23. 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.