Coordination between Logical Agents

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. In this paper we suppose an agent that has a knowledge base written in logic programming and sets of beliefs under the answer set semantics. We then consider the following two problems: given two logic programs P_1 and P_2, which have the sets of answer sets $\mathcal{AS}(P_1)$ and $\mathcal{AS}(P_2)$, respectively; (i) find a program Q which has the set of answer sets such that $\mathcal{AS}(Q) = \mathcal{AS}(P_1) \cup \mathcal{AS}(P_2)$; (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 satisfying the condition (i) is called *generous coordination* of P_1 and P_2; and R satisfying (ii) is called *rigorous coordination* of P_1 and P_2. Generous coordination retains all of the original belief sets of each agent, but admits the introduction of additional belief sets of the other agent. By contrast, rigorous coordination forces each agent to give up some belief sets, but the result remains within the original belief sets for each agent. We provide methods for constructing these two types of coordination and discuss their properties.

1 Introduction

In multi-agent systems different agents may have different sets of beliefs, and agents negotiate and accommodate themselves to reach acceptable agreements. We call a process of forming such agreements between agents *coordination*. The problem is how to settle an agreement acceptable to each agent. The outcome of coordination is required to be consistent and is desirable to retain original information of each agent as much as possible.

Suppose an agent that has a knowledge base as a logic program whose semantics is given as the collection of *answer sets* [7]. Answer sets represent sets of literals corresponding to beliefs which can be built by a rational reasoner on the basis of a program [2]. An agent may have (conflicting) alternative sets of beliefs, which are represented by multiple answer sets of a program. Different agents have different collections of answer sets in general. We then capture coordination between two agents as the problem of finding a new program which has the meaning balanced between two programs. Consider, for instance, a logic program P_1 which has two answer sets S_1 and S_2; and another logic program

P_2 which has two answer sets S_2 and S_3. Then, we want to find a new program which is a result of coordination between P_1 and P_2. In this paper, we consider two different solutions: one is a program Q which has three answer sets S_1, S_2, and S_3; the other is a program R which has the single answer set S_2.

These two solutions provide different types of coordination — the first one retains all of the original belief sets of each agent, but admits the introduction of additional belief sets of the other agent. By contrast, the second one forces each agent to give up some belief sets, but the result remains within the original belief sets for each agent. These two types of coordination occur in real life. For instance, suppose the following scenario: to decide the Academy Award of Best Pictures, each member of the Academy nominates films. Now there are three members — p_1, p_2, and p_3, and each member can nominate at most two films: p_1 nominates f_1 and f_2, p_2 nominates f_2 and f_3, and p_3 nominates f_2. At this moment, three nominees f_1, f_2, and f_3 are fixed. The situation is represented by three programs:

$$\begin{aligned} P_1: &\quad f_1 \,;\, f_2 \leftarrow, \\ P_2: &\quad f_2 \,;\, f_3 \leftarrow, \\ P_3: &\quad f_2 \leftarrow, \end{aligned}$$

where ";" represents disjunction. Here, P_1 has two answer sets: $\{f_1\}$ and $\{f_2\}$; P_2 has two answer sets: $\{f_2\}$ and $\{f_3\}$; P_3 has the single answer set: $\{f_2\}$. The three nominees correspond to the answer sets: $\{f_1\}$, $\{f_2\}$, and $\{f_3\}$. A program having these three answer sets is the first type of coordination. After final voting, the film f_2 is supported by three members and becomes the winner of the Award. That is, the winner is represented by the answer set $\{f_2\}$. A program having this single answer set is the second type of coordination. Thus, these two types of coordination happen in different situations, and it is meaningful to develop computational logic for these coordination between agents.

The problem is then how to build a program which realizes such coordination. Formally, the problems considered in this paper are described as follows.

Given: two programs P_1 and P_2;

Find: (1) a program Q satisfying $\mathcal{AS}(Q) = \mathcal{AS}(P_1) \cup \mathcal{AS}(P_2)$;

(2) a program R satisfying $\mathcal{AS}(R) = \mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)$,

where $\mathcal{AS}(P)$ represents the set of answer sets of a program P. The program Q satisfying (1) is called *generous coordination* of P_1 and P_2; and the program R satisfying (2) is called *rigorous coordination* of P_1 and P_2. We develop methods for computing these two types of coordination and verify the results.

The rest of this paper is organized as follows. Section 2 presents definitions and terminologies used in this paper. Section 3 introduces a framework of coordination between logic programs. Section 4 provides methods for computing coordination and addresses their properties. Section 5 discusses related issues and Section 6 summarizes the paper.

2 Preliminaries

In this paper, we suppose an agent that has a knowledge base written in logic programming. An agent is then identified with its logic program and we use those terms interchangeably throughout the paper.

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, $head(r)$, $body^+(r)$, $body^-(r)$, and $not\_body^-(r)$ denote the sets of (NAF-)literals $\{L_1, \ldots, L_l\}$, $\{L_{l+1}, \ldots, L_m\}$, $\{L_{m+1}, \ldots, L_n\}$, and $\{not\,L_{m+1}, \ldots, not\,L_n\}$, respectively. A disjunction of literals and a conjunction of (NAF-)literals in a rule are identified with its corresponding sets of (NAF-)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 P is *NAF-free* if $body^-(r) = \emptyset$ for any rule r in P. A program with variables is semantically identified with its ground instantiation, and we handle propositional and ground programs throughout the paper.

The semantics of EDPs is given by the *answer set semantics* [7]. 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. When $body^+(r) \subseteq S$ (resp. $head(r) \cap S \neq \emptyset$), it is also written as $S \models body^+(r)$ (resp. $S \models head(r)$).

Let P be an NAF-free EDP. Then, a set $S(\subset Lit)$ is a *(consistent) answer set* of P if S is a minimal set such that

1. S satisfies every rule from the ground instantiation of P,
2. S does not contain a pair of complementary literals L and $\neg L$ for any $L \in Lit$.

Next, let P be any EDP and $S \subset Lit$. For every rule r in the ground instantiation of P, the rule $r^S:\; head(r) \leftarrow body^+(r)$ is included in the *reduct* P^S if $body^-(r) \cap S = \emptyset$. Then, S is an *answer set* of P if S is an answer set of P^S. An EDP 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 P is *consistent* if it has a consistent answer set. In this paper, we assume that a program is consistent unless stated otherwise.

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, $skp(P) \subseteq crd(P)$ holds for any P. Two programs

P_1 and P_2 are said to be *AS-combinable* if every set in $\mathcal{AS}(P_1) \cup \mathcal{AS}(P_2)$ is minimal under set inclusion.

Example 2.1. Given two programs:

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

where $\mathcal{AS}(P_1) = \{\{p, q\}\}$ and $\mathcal{AS}(P_2) = \{\{p\}, \{q\}\}$. Then, $crd(P_1) = skp(P_1) = \{p, q\}$; $crd(P_2) = \{p, q\}$ and $skp(P_2) = \emptyset$. P_1 and P_2 are not AS-combinable because the set $\{p, q\}$ is not minimal in $\mathcal{AS}(P_1) \cup \mathcal{AS}(P_2)$.

Technically, when two programs P_1 and P_2 are not AS-combinable, we can make them AS-combinable by introducing the rule $\overline{L} \leftarrow not\, L$ for every $L \in Lit$ to each program, where $\overline{L}$ is a newly introduced atom associated uniquely with each L.

Example 2.2. In the above example, put $P_1' = P_1 \cup Q$ and $P_2' = P_2 \cup Q$ with

$$\begin{aligned} Q: \quad & \overline{p} \leftarrow not\, p, \\ & \overline{q} \leftarrow \ not\, q\,. \end{aligned}$$

Then, $\mathcal{AS}(P_1') = \{\{p, q\}\}$ and $\mathcal{AS}(P_2') = \{\{p, \overline{q}\}, \{\overline{p}, q\}\}$, so P_1' and P_2' are AS-combinable.

3 Coordination between Programs

Given two programs, coordination provides a program which is a reasonable compromise between agents. In this section, we introduce two different types of coordination under the answer set semantics.

Definition 3.1. Let P_1 and P_2 be two programs. A program Q satisfying the condition $\mathcal{AS}(Q) = \mathcal{AS}(P_1) \cup \mathcal{AS}(P_2)$ is called *generous coordination* of P_1 and P_2; a program R satisfying the condition $\mathcal{AS}(R) = \mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)$ is called *rigorous coordination* of P_1 and P_2.

Generous coordination retains all of the answer sets of each agent, but admits the introduction of additional answer sets of the other agent. By contrast, rigorous coordination forces each agent to give up some answer sets, but the result remains within the original answer sets for each agent.

Technically, generous coordination requires two programs P_1 and P_2 to be AS-combinable, since answer sets of Q are all minimal. Thus, when we consider generous coordination between two programs, we assume them to be AS-combinable. Generous coordination between programs that are not AS-combinable is possible by making them AS-combinable in advance using the program transformation presented in Section 2.

Definition 3.2. For two programs P_1 and P_2, let Q be a result of generous coordination, and R a result of rigorous coordination. We say that generous (resp. rigorous) coordination *succeeds* if $\mathcal{AS}(Q) \neq \emptyset$ (resp. $\mathcal{AS}(R) \neq \emptyset$); otherwise, it *fails*.

Generous coordination always succeeds whenever both P_1 and P_2 are consistent. On the other hand, when $\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2) = \emptyset$, rigorous coordination fails as two agents have no common belief sets. Note that generous coordination may produce a collection of answer sets which contradict with one another. But this does not cause any problem as a collection of answer sets represents (conflicting) alternative belief sets of each agent.

As we assume consistent programs, the next result holds by the definition.

Proposition 3.1 *When generous/rigorous coordination of two programs succeeds, the result of coordination is consistent.*

Coordination changes the consequences of credulous/skeptical reasoning by each agent.

Proposition 3.2 *Let P_1 and P_2 be two programs.*

1. *If Q is a result of generous coordination,*
   (a) $\mathit{crd}(Q) = \mathit{crd}(P_1) \cup \mathit{crd}(P_2)$ *;*
   (b) $\mathit{skp}(Q) = \mathit{skp}(P_1) \cap \mathit{skp}(P_2)$ *;*
   (c) $\mathit{crd}(Q) \supseteq \mathit{crd}(P_i)$ *for* $i = 1, 2$ *;*
   (d) $\mathit{skp}(Q) \subseteq \mathit{skp}(P_i)$ *for* $i = 1, 2$.
2. *If R is a result of rigorous coordination,*
   (a) $\mathit{crd}(R) \subseteq \mathit{crd}(P_1) \cup \mathit{crd}(P_2)$ *;*
   (b) $\mathit{skp}(R) \supseteq \mathit{skp}(P_1) \cup \mathit{skp}(P_2)$ *if* $\mathcal{AS}(R) \neq \emptyset$ *;*
   (c) $\mathit{crd}(R) \subseteq \mathit{crd}(P_i)$ *for* $i = 1, 2$ *;*
   (d) $\mathit{skp}(R) \supseteq \mathit{skp}(P_i)$ *for* $i = 1, 2$ *if* $\mathcal{AS}(R) \neq \emptyset$.

Proof. 1.(a) A literal L is included in an answer set in $\mathcal{AS}(P_1) \cup \mathcal{AS}(P_2)$ iff L is included in an answer set in $\mathcal{AS}(P_1)$ or included in an answer set in $\mathcal{AS}(P_2)$. (b) L is included in every answer set in $\mathcal{AS}(P_1) \cup \mathcal{AS}(P_2)$ iff L is included in every answer set in $\mathcal{AS}(P_1)$ and also included in every answer set in $\mathcal{AS}(P_2)$. The results of (c) and (d) hold by (a) and (b), respectively.

2.(a) If L is included in an answer set in $\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)$, L is included in an answer set in $\mathcal{AS}(P_i)$ $(i = 1, 2)$. (b) If L is included in every answer set of either P_1 or P_2, L is included in every answer set in $\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)$ if the intersection is nonempty. The results of (c) and (d) hold by (a) and (b), respectively. $\square$

Example 3.1. Let $\mathcal{AS}(P_1) = \{\{a, b, c\}, \{b, c, d\}\}$ and $\mathcal{AS}(P_2) = \{\{b, c, d\}, \{c, e\}\}$, where $\mathit{crd}(P_1) = \{ a, b, c, d \}$, $\mathit{skp}(P_1) = \{ b, c \}$, $\mathit{crd}(P_2) = \{ b, c, d, e \}$, and $\mathit{skp}(P_2) = \{ c \}$. Generous coordination Q of P_1 and P_2 has the answer sets $\mathcal{AS}(Q) = \{\{a, b, c\}, \{b, c, d\}, \{c, e\}\}$ where $\mathit{crd}(Q) = \{ a, b, c, d, e \}$ and $\mathit{skp}(Q) = \{c\}$. Rigorous coordination R has the answer sets $\mathcal{AS}(R) = \{\{b, c, d\}\}$ where $\mathit{crd}(R) = \mathit{skp}(R) = \{ b, c, d \}$. The above relations are verified for these sets.

Generous coordination merges credulous consequences of P_1 and P_2, while restricts skeptical consequences to those that are common between two programs. As a result, it increases credulous consequences and decreases skeptical consequences. This reflects the situation that accepting opinions of the other agent increases alternative choices while weakening the original argument of each agent. By contrast, rigorous coordination reduces credulous consequences, but increases skeptical consequences in general. This reflects the situation that excluding opinions of the other agent costs abandoning some of one's alternative beliefs, which results in strengthening some original argument of each agent.

Definition 3.3. For two programs P_1 and P_2, let Q be a result of generous coordination, and R a result of rigorous coordination. When $\mathcal{AS}(Q) = \mathcal{AS}(P_1)$ (resp. $\mathcal{AS}(R) = \mathcal{AS}(P_1)$), P_1 *dominates* P_2 under generous (resp. rigorous) coordination.

Proposition 3.3 *Let P_1 and P_2 be two programs. When $\mathcal{AS}(P_1) \subseteq \mathcal{AS}(P_2)$, P_2 dominates P_1 under generous coordination, and P_1 dominates P_2 under rigorous coordination.*

When P_2 dominates P_1 under generous coordination, we can easily have a result of generous coordination as $Q = P_2$. Similarly, when P_1 dominates P_2 under rigorous coordination, a result of rigorous coordination becomes $R = P_1$.

In cases where one agent dominates the other one, or when coordination fails, the results of coordination are trivial and uninteresting. Then, the problem of interest is the cases where $\mathcal{AS}(P_1) \not\subseteq \mathcal{AS}(P_2)$ and $\mathcal{AS}(P_2) \not\subseteq \mathcal{AS}(P_1)$ for computing generous/rigorous coordination; and $\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2) \neq \emptyset$ for computing rigorous coordination. In the next section, we present methods for computing these two coordination.

4 Computing Coordination

4.1 Computing Generous Coordination

We first present a method of computing generous coordination between two programs.

Definition 4.1. Given two programs P_1 and P_2,

$$P_1 \oplus P_2 = \{\, \mathit{head}(r_1)\,;\, \mathit{head}(r_2) \leftarrow \mathit{body}_*(r_1),\, \mathit{body}_*(r_2) \mid r_1 \in P_1,\, r_2 \in P_2 \,\},$$

where $\mathit{head}(r_1)\,;\,\mathit{head}(r_2)$ is the disjunction of $\mathit{head}(r_1)$ and $\mathit{head}(r_2)$, $\mathit{body}_*(r_1) = \mathit{body}(r_1) \setminus \{\, \mathit{not}\, L \mid L \in T \setminus S \,\}$ and $\mathit{body}_*(r_2) = \mathit{body}(r_2) \setminus \{\, \mathit{not}\, L \mid L \in S \setminus T \,\}$ for any $S \in \mathcal{AS}(P_1)$ and $T \in \mathcal{AS}(P_2)$.

The program $P_1 \oplus P_2$ is a collection of rules which are obtained by combining a rule of P_1 and a rule of P_2 in every possible way. In $\mathit{body}_*(r_1)$ every NAF-literal $\mathit{not}\, L$ such that $L \in T \setminus S$ is dropped because the existence of this may prevent the derivation of some literal in $\mathit{head}(r_2)$ after combination.

Example 4.1. Consider two programs:

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

where $\mathcal{AS}(P_1) = \{\{p\}, \{q\}\}$ and $\mathcal{AS}(P_2) = \{\{\neg p\}\}$. Then, $P_1 \oplus P_2$ becomes

$$\begin{aligned} & p\,;\, \neg p \leftarrow not\, q, \\ & q\,;\, \neg p \leftarrow not\, p. \end{aligned}$$

Note that $not\, p$ from the rule of P_2 is dropped in the resulting rules because of the existence of $\{p\}$ in $\mathcal{AS}(P_1)$.

By the definition, $P_1 \oplus P_2$ 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 in P.

The program $P_1 \oplus P_2$ generally contains useless or redundant literals/rules, and the following program transformations are helpful to simplify the program.

- (elimination of tautologies: TAUT)
  Delete a rule r from a program if $head(r) \cap body^+(r) \neq \emptyset$.
- (elimination of contradictions: CONTRA)
  Delete a rule r from a program if $body^+(r) \cap body^-(r) \neq \emptyset$.
- (elimination of non-minimal rules: NONMIN)
  Delete a rule r from a program if there is another rule r' in the program such that $head(r') \subseteq head(r)$, $body^+(r') \subseteq body^+(r)$ and $body^-(r') \subseteq body^-(r)$.
- (merging duplicated literals: DUPL)
  A disjunction $(L; L)$ appearing in $head(r)$ is merged into L, and a conjunction (L, L) or $(not\, L,\, not\, L)$ appearing in $body(r)$ is merged into L or $not\, L$, respectively.

These program transformations all preserve the answer sets of an EDP [3].

Example 4.2. Given two programs:

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

$P_1 \oplus P_2$ becomes

$$\begin{aligned} & p\,;\, p \leftarrow q,\, not\, q, \\ & p\,;\, q \leftarrow q,\, r, \\ & p\,;\, r \leftarrow not\, q, \\ & r\,;\, q \leftarrow r. \end{aligned}$$

The first rule is deleted by CONTRA, the second rule and the fourth rule are deleted by TAUT. After such elimination, the resulting program contains the third rule only.

Now we show that $P_1 \oplus P_2$ realizes generous coordination of P_1 and P_2.

Lemma 4.1 *Let P_1 and P_2 be two NAF-free AS-combinable programs. Then, S is an answer set of $P_1 \oplus P_2$ iff S is an answer set of either P_1 or P_2.*

Proof. Suppose that S is an answer set of P_1. Then, S satisfies any rule $head(r_1) \leftarrow body(r_1)$ in P_1, thereby satisfies any rule $head(r_1); head(r_2) \leftarrow body(r_1), body(r_2)$ in $P_1 \oplus P_2$. (Note: $body_*(r_i) = body(r_i)$ for NAF-free programs.) To see that S is an answer set of $P_1 \oplus P_2$, suppose that there is a minimal set $T \subset S$ which satisfies every rule in $P_1 \oplus P_2$. Since S is an answer set of P_1, there is a rule r'_1 in P_1 which is not satisfied by T. For this rule, $T \not\models head(r'_1)$ and $T \models body(r'_1)$ hold. Then, for any rule $head(r'_1); head(r_2) \leftarrow body(r'_1), body(r_2)$ in $P_1 \oplus P_2$, $T \models head(r_2)$ or $T \not\models body(r_2)$. Since every rule in P_2 is combined with r'_1, it holds that $T \models head(r_2)$ or $T \not\models body(r_2)$ for every r_2 in P_2. Then, T satisfies P_2. As P_2 is consistent, it has an answer set $T' \subseteq T$. This contradicts the assumption that P_1 and P_2 are AS-combinable, i.e., $T' \not\subset S$. Hence, S is an answer set of $P_1 \oplus P_2$. The case that S is an answer set of P_2 is proved in the same manner.

Conversely, suppose that S is an answer set of $P_1 \oplus P_2$. Then, S satisfies any rule $head(r_1); head(r_2) \leftarrow body(r_1), body(r_2)$ in $P_1 \oplus P_2$. Then $S \models body(r_1), body(r_2)$ implies $S \models head(r_1); head(r_2)$. If $S \not\models head(r_1)$ for some rule $r_1 \in P_1$, $S \models head(r_2)$ for any $r_2 \in P_2$. Then, $S \models body(r_2)$ implies $S \models head(r_2)$ for any $r_2 \in P_2$, so that S satisfies every rule in P_2. Else if $S \not\models head(r_2)$ for some rule $r_2 \in P_2$, $S \models head(r_1)$ for any $r_1 \in P_1$. Then, $S \models body(r_1)$ implies $S \models head(r_1)$ for any $r_1 \in P_1$, so that S satisfies every rule in P_1. Else if $S \models head(r_1)$ for every $r_1 \in P_1$ and $S \models head(r_2)$ for every $r_2 \in P_2$, S satisfies both P_1 and P_2. Thus, in every case S satisfies either P_1 or P_2. Suppose that S satisfies P_1 but it is not an answer set of P_1. Then, there is an answer set T of P_1 such that $T \subset S$. By the if-part, T becomes an answer set of $P_1 \oplus P_2$. This contradicts the assumption that S is an answer set of $P_1 \oplus P_2$. Similar argument is applied when S satisfies P_2. □

Theorem 4.2. *Let P_1 and P_2 be two AS-combinable programs. Then, $\mathcal{AS}(P_1 \oplus P_2) = \mathcal{AS}(P_1) \cup \mathcal{AS}(P_2)$.*

Proof. Suppose $S \in \mathcal{AS}(P_1)$. Then, S is an answer set of P_1^S, so that S is an answer set of $P_1^S \oplus P_2^T$ for any $T \in \mathcal{AS}(P_2)$ (Lemma 4.1). (Note: as P_1 and P_2 are AS-combinable, the reducts P_1^S and P_2^T are also AS-combinable.) For any rule $head(r_1); head(r_2) \leftarrow body^+(r_1), body^+(r_2)$ in $P_1^S \oplus P_2^T$, it holds that $body^-(r_1) \cap S = body^-(r_2) \cap T = \emptyset$. On the other hand, for any rule $head(r_1); head(r_2) \leftarrow body_*(r_1), body_*(r_2)$ in $P_1 \oplus P_2$, $head(r_1); head(r_2) \leftarrow body^+(r_1), body^+(r_2)$ is in $(P_1 \oplus P_2)^S$ iff $(body^-(r_1) \setminus \{ L \mid L \in T \setminus S' \}) \cap S = \emptyset$ and $(body^-(r_2) \setminus \{ L \mid L \in S' \setminus T \}) \cap S = \emptyset$ for any $S' \in \mathcal{AS}(P_1)$ and $T \in \mathcal{AS}(P_2)$. Here it holds that $(body^-(r_1) \setminus \{ L \mid L \in T \setminus S' \}) \cap S \subseteq body^-(r_1) \cap S$ and $(body^-(r_2) \setminus \{ L \mid L \in$

$S' \setminus T\,\}) \cap S \subseteq \mathit{body}^-(r_2) \cap T \cap S \subseteq \mathit{body}^-(r_2) \cap T$. Hence, $P_1^S \oplus P_2^T \subseteq (P_1 \oplus P_2)^S$. Suppose any rule $\mathit{head}(r_1); \mathit{head}(r_2) \leftarrow \mathit{body}^+(r_1), \mathit{body}^+(r_2)$ in $(P_1 \oplus P_2)^S \setminus (P_1^S \oplus P_2^T)$. Since S satisfies any rule r_1 in P_1, $S \models \mathit{body}^+(r_1), \mathit{body}^+(r_2)$ implies $S \models \mathit{head}(r_1); \mathit{head}(r_2)$. Thus, the answer set S of $P_1^S \oplus P_2^T$ satisfies every rule in $(P_1 \oplus P_2)^S \setminus (P_1^S \oplus P_2^T)$. By $P_1^S \oplus P_2^T \subseteq (P_1 \oplus P_2)^S$, S becomes an answer set of $(P_1 \oplus P_2)^S$ and $S \in \mathcal{AS}(P_1 \oplus P_2)$. The case of $S \in \mathcal{AS}(P_2)$ is proved in the same manner.

Conversely, suppose $S \in \mathcal{AS}(P_1 \oplus P_2)$. Then, S satisfies any rule $\mathit{head}(r_1); \mathit{head}(r_2) \leftarrow \mathit{body}_*(r_1), \mathit{body}_*(r_2)$ in $P_1 \oplus P_2$, so $S \models \mathit{body}_*(r_1), \mathit{body}_*(r_2)$ implies $S \models \mathit{head}(r_1); \mathit{head}(r_2)$. If $S \not\models \mathit{head}(r_1)$ for some rule $r_1 \in P_1$, $S \models \mathit{head}(r_2)$ for any $r_2 \in P_2$. Then, $S \models \mathit{body}_*(r_2)$ implies $S \models \mathit{head}(r_2)$ for any $r_2 \in P_2$, so $S \models \mathit{head}(r_2)$ or $S \not\models \mathit{body}_*(r_2)$. As $S \not\models \mathit{body}_*(r_2)$ implies $S \not\models \mathit{body}(r_2)$, it holds that $S \models \mathit{head}(r_2)$ or $S \not\models \mathit{body}(r_2)$ for any $r_2 \in P$. Hence, S satisfies every rule in P_2. Else if $S \not\models \mathit{head}(r_2)$ for some rule $r_2 \in P_2$, it is shown in a similar manner that S satisfies every rule in P_1. Else if $S \models \mathit{head}(r_1)$ for every $r_1 \in P_1$ and $S \models \mathit{head}(r_2)$ for every $r_2 \in P_2$, S satisfies both P_1 and P_2. Thus, in every case S satisfies either P_1 or P_2. Suppose that S satisfies P_1 but it is not an answer set of P_1. Then, there is an answer set T of P_1 such that $T \subset S$. By the if-part, T becomes an answer set of $P_1 \oplus P_2$. This contradicts the assumption that S is an answer set of $P_1 \oplus P_2$. Similar argument is applied when S satisfies P_2. $\square$

Example 4.3. In Example 4.1, $\mathcal{AS}(P_1 \oplus P_2) = \{\{p\}, \{q\}, \{\neg p\}\}$, thereby $\mathcal{AS}(P_1 \oplus P_2) = \mathcal{AS}(P_1) \cup \mathcal{AS}(P_2)$.

4.2 Computing Rigorous Coordination

Next we present a method of computing rigorous coordination between two programs.

Definition 4.2. Given two programs P_1 and P_2,

$$P_1 \otimes P_2 = \bigcup_{S \in \mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)} R(P_1, S) \,\cup\, R(P_2, S),$$

where $\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2) \neq \emptyset$ and

$$R(P, S) = \{\, \mathit{head}(r) \cap S \leftarrow \mathit{body}(r),\ \mathit{not}\ (\mathit{head}(r) \setminus S) \ \mid\ r \in P \text{ and } r^S \in P^S \,\}$$

and $\quad \mathit{not}\ (\mathit{head}(r) \setminus S) = \{\, \mathit{not}\, L \ \mid\ L \in \mathit{head}(r) \setminus S \,\}$.

When $\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2) = \emptyset$, $P_1 \otimes P_2$ is undefined.[3]

Intuitively, the program $P_1 \otimes P_2$ is a collection of rules which may be used for constructing answer sets that are common between P_1 and P_2. In $R(P, S)$ any literal in $\mathit{head}(r)$ which does not contribute to the construction of the answer set S is shifted to the body as NAF-literals. $P_1 \otimes P_2$ may contain redundant rules, which are eliminated using program transformations given in the previous subsection.

[3] Technically, $P_1 \otimes P_2$ is set as $\{\, p \leftarrow \mathit{not}\, p \,\}$ for any atom p.

Example 4.4. Consider two programs:

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

where $\mathcal{AS}(P_1) = \{\{p\},\{q\},\{r\}\}$, $\mathcal{AS}(P_2) = \{\{p\},\{q\},\{\neg r\}\}$, and $\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2) = \{\{p\},\{q\}\}$. Then, $P_1 \otimes P_2$ becomes

$$\begin{aligned} & p \leftarrow \mathit{not}\, q,\ \mathit{not}\, r, \\ & q \leftarrow \mathit{not}\, p,\ \mathit{not}\, r, \\ & p \leftarrow \mathit{not}\, r,\ \mathit{not}\, q,\ \mathit{not}\, \neg r, \\ & q \leftarrow \mathit{not}\, r,\ \mathit{not}\, p,\ \mathit{not}\, \neg r. \end{aligned}$$

Here, the third and the fourth rules can be eliminated by NONMIN.

By the definition, $P_1 \otimes P_2$ is computed in time $(|P_1| + |P_2|) \times |\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)|$ where $|\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)|$ represents the number of answer sets in $\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)$.

$P_1 \otimes P_2$ realizes rigorous coordination of P_1 and P_2.

Lemma 4.3 *Let P be a program. Then, S is an answer set of P iff S is an answer set of $R(P,S)$.*

Proof. S is an answer set of P iff S is an answer set of P^S
iff S is a minimal set such that $\mathit{body}^+(r) \subseteq S$ implies $\mathit{head}(r) \cap S \neq \emptyset$ for every rule $\mathit{head}(r) \leftarrow \mathit{body}^+(r)$ in P^S $(*)$. By the definition of $R(P,S)$, the rule $\mathit{head}(r) \leftarrow \mathit{body}^+(r)$ is in P^S iff the corresponding rule $\mathit{head}(r) \cap S \leftarrow \mathit{body}^+(r)$ is in $R(P,S)^S$ (because $\mathit{body}^-(r) \cap S = \emptyset$ and $(\mathit{head}(r) \setminus S) \cap S = \emptyset$). Hence, the statement $(*)$ holds iff S is a minimal set such that $\mathit{body}^+(r) \subseteq S$ implies $\mathit{head}(r) \cap S \neq \emptyset$ for every rule $\mathit{head}(r) \cap S \leftarrow \mathit{body}^+(r)$ in $R(P,S)^S$
iff S is a minimal set which satisfies every rule $\mathit{head}(r) \cap S \leftarrow \mathit{body}^+(r)$ in $R(P,S)^S$
iff S is an answer set of $R(P,S)$. □

Theorem 4.4. *Let P_1 and P_2 be two programs. Then, $\mathcal{AS}(P_1 \otimes P_2) = \mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)$.*

Proof. Suppose $S \in \mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)$. Then, S satisfies any rule $\mathit{head}(r) \leftarrow \mathit{body}(r)$ in P_1 and P_2, so that S satisfies the corresponding rules $\mathit{head}(r) \cap T \leftarrow \mathit{body}(r),\ \mathit{not}\,(\mathit{head}(r) \setminus T)$ in $R(P_1,T) \cup R(P_2,T)$ for any $T \in \mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)$. Thus, S satisfies $P_1 \otimes P_2$. Suppose that S is not an answer set of $P_1 \otimes P_2$. Then, there is a minimal set $U \subset S$ which satisfies every rule in $P_1 \otimes P_2$. In this case, U satisfies $R(P_1,S)$. By Lemma 4.3, however, S is a minimal set which satisfies $R(P_1,S)$. Contradiction. Hence, S is an answer set of $P_1 \otimes P_2$.

Conversely, suppose $S \in \mathcal{AS}(P_1 \otimes P_2)$. Then, S is a minimal set which satisfies every rule $head(r) \cap T \leftarrow body(r),\ not\,(head(r) \setminus T)$ in $R(P_1, T) \cup R(P_2, T)$ for any $T \in \mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)$. By Lemma 4.3, T is also a minimal set which satisfies both $R(P_1, T)$ and $R(P_2, T)$, so that there is a literal $L \in S \setminus T$ and a literal $M \in T \setminus S$. However, any rule in $R(P_1, T) \cup R(P_2, T)$ has the head $head(r) \cap T$, so that no literal $L \in S \setminus T$ is included in the head. Thus, L is not included in the answer set S, thereby $S \setminus T = \emptyset$. As both T and S are minimal, $T \setminus S = \emptyset$. Hence, $T = S$ and $S \in \mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)$. □

Example 4.5. In Example 4.4, $\mathcal{AS}(P_1 \otimes P_2) = \{\{p\}, \{q\}\}$, thereby $\mathcal{AS}(P_1 \otimes P_2) = \mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)$.

4.3 Algebraic Properties

In this subsection, we provide properties of the operations $\oplus$ and $\otimes$.

Proposition 4.5 *For programs P_1, P_2, and P_3, the operations $\oplus$ and $\otimes$ have the following properties:*

(i) $P_1 \oplus P_2 = P_2 \oplus P_1$ *and* $P_1 \otimes P_2 = P_2 \otimes P_1$*;*
(ii) $(P_1 \oplus P_2) \oplus P_3 = P_1 \oplus (P_2 \oplus P_3)$ *if P_1, P_2 and P_3 are NAF-free;*
(iii) $(P_1 \otimes P_2) \otimes P_3 = P_1 \otimes (P_2 \otimes P_3)$.

Proof. The results of (i) and (ii) are straightforward. To see (iii), $\mathcal{AS}(P_1 \otimes P_2) = \mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)$ holds by Theorem 4.4. Then, both $(P_1 \otimes P_2) \otimes P_3$ and $P_1 \otimes (P_2 \otimes P_3)$ consist of rules in $R(P_1, S) \cup R(P_2, S) \cup R(P_3, S)$ for every $S \in \mathcal{AS}(P_1) \cap \mathcal{AS}(P_2) \cap \mathcal{AS}(P_3)$. □

The operation $\oplus$ is not associative in general when programs contain NAF, but it holds the relation $\mathcal{AS}((P_1 \oplus P_2) \oplus P_3) = \mathcal{AS}(P_1 \oplus (P_2 \oplus P_3))$. $\oplus$ is also *idempotent*, $P \oplus P = P$ if NONMIN and DUPL are applied to $P \oplus P$ and P. $\otimes$ is not idempotent but the relation $\mathcal{AS}(P \otimes P) = \mathcal{AS}(P)$ holds. By the definition, $P \otimes P$ has the effect of extracting rules used for constructing answer sets of P.

By Proposition 4.5, when rigorous coordination are done among more than two agents, the order of computing coordination does not affect the result of final outcome. This is also the case for generous coordination when programs are NAF-free.

Two types of coordination are mixed among agents. In this case, the absorption laws and the distribution laws do not hold in general, i.e.,

$$\begin{aligned}
&P_1 \oplus (P_1 \otimes P_2) \neq P_1 \ \text{ and } \ P_1 \otimes (P_1 \oplus P_2) \neq P_1;\\
&P_1 \oplus (P_2 \otimes P_3) \neq (P_1 \oplus P_2) \otimes (P_1 \oplus P_3) \ \text{ and}\\
&P_1 \otimes (P_2 \oplus P_3) \neq (P_1 \otimes P_2) \oplus (P_1 \otimes P_3),
\end{aligned}$$

Note that programs are generally different, but the following relations hold by the definitions:

$$\begin{aligned}
&\mathcal{AS}(P_1 \oplus (P_1 \otimes P_2)) = \mathcal{AS}(P_1 \otimes (P_1 \oplus P_2)) = \mathcal{AS}(P_1),\\
&\mathcal{AS}(P_1 \oplus (P_2 \otimes P_3)) = \mathcal{AS}((P_1 \oplus P_2) \otimes (P_1 \oplus P_3)),\\
&\mathcal{AS}(P_1 \otimes (P_2 \oplus P_3)) = \mathcal{AS}((P_1 \otimes P_2) \oplus (P_1 \otimes P_3)).
\end{aligned}$$

5 Discussion

When a set of answer sets is given, it is not difficult to construct a program which has exactly those answer sets. Given a set of answer sets $\{S_1, \ldots, S_m\}$, first compute the disjunctive normal form: $S_1 \vee \cdots \vee S_m$, then convert it into the conjunctive normal form: $R_1 \wedge \cdots \wedge R_n$. The set of facts $\{R_1, \ldots, R_n\}$ then has the answer sets $\{S_1, \ldots, S_m\}$. This technique is also used for computing coordination between programs. For instance, suppose two programs:

$$\begin{aligned} P_1 :\ & \mathit{sweet} \leftarrow \mathit{strawberry}, \\ & \mathit{strawberry} \leftarrow, \\ P_2 :\ & \mathit{red} \leftarrow \mathit{strawberry}, \\ & \mathit{strawberry} \leftarrow, \end{aligned}$$

where $\mathcal{AS}(P_1) = \{\{\mathit{sweet}, \mathit{strawberry}\}\}$ and $\mathcal{AS}(P_2) = \{\{\mathit{red}, \mathit{strawberry}\}\}$.

To get generous coordination which has the answer sets $\mathcal{AS}(P_1) \cup \mathcal{AS}(P_2)$, taking the DNF of each answer set produces

$$(\mathit{sweet} \wedge \mathit{strawberry}) \ \vee \ (\mathit{red} \wedge \mathit{strawberry}).$$

Converting it into the CNF, it becomes

$$(\mathit{sweet} \vee \mathit{red}) \wedge \mathit{strawberry}.$$

As a result, the set of facts

$$\begin{aligned} Q :\ & \mathit{sweet}\,;\, \mathit{red} \leftarrow, \\ & \mathit{strawberry} \leftarrow \end{aligned}$$

is a program which is generous coordination of P_1 and P_2. On the other hand, the program $P_1 \oplus P_2$ becomes

$$\begin{aligned} & \mathit{sweet}\,;\, \mathit{red} \leftarrow \mathit{strawberry}, \\ & \mathit{strawberry} \leftarrow, \end{aligned}$$

after eliminating duplicated literals and redundant rules.

These two programs have the same meaning but have different syntax. Then, a question is which one is more preferable as a result of coordination? Our answer is $P_1 \oplus P_2$. The intuition behind this selection is that we would like to include as much information as possible from the original programs. Comparing Q with $P_1 \oplus P_2$, information of dependency between *sweet* (or *red*) and *strawberry* is lost in Q.[4] Generally speaking, if there exist different candidates for coordination between two programs, a program which is syntactically closer to the original ones is preferred. Then, a question is how to measure such "syntactical closeness" between programs? One solution we have in mind is, as illustrated above, using

[4] Technically, the program Q is obtained by unfolding rules in $P_1 \oplus P_2$ [3, 11].

dependency relations between literals. We prefer a result of coordination which inherits dependency relations from the original programs as much as possible.

More precisely, suppose the *dependency graph* of a program P in which each node represents a ground literal and there is a directed edge from L_1 to L_2 (we say L_1 *depends on* L_2) iff there is a ground rule in P such that L_1 appears in the head and L_2 appears in the body of the rule. Let (L_1, L_2) be a pair of ground literals such that L_1 depends on L_2 in the dependency graph of a program. Let $\delta(P)$ be the collection of such pairs in P. For two programs P_1 and P_2, suppose that two different programs P_3 and P_4 are obtained as candidates for coordination. Then, we say that P_3 is *preferable* to P_4 if

$$\Delta(\delta(P_3), \delta(P_1) \cup \delta(P_2)) \subset \Delta(\delta(P_4), \delta(P_1) \cup \delta(P_2)),$$

where $\Delta(S,T)$ represents the symmetric difference between two sets S and T, i.e., $(S \setminus T) \cup (T \setminus S)$. Applying to the above example, $\delta(P_1) = \{(sweet, strawberry)\}$, $\delta(P_2) = \{(red, strawberry)\}$, $\delta(Q) = \emptyset$, and $\delta(P_1 \oplus P_2) = \{(sweet, strawberry), (red, strawberry)\}$. Then, $\Delta(\delta(P_1 \oplus P_2), \delta(P_1) \cup \delta(P_2)) \subset \Delta(\delta(Q), \delta(P_1) \cup \delta(P_2))$, so we conclude that $P_1 \oplus P_2$ is preferable to Q. Further elaboration would be considered to reflect syntactical closeness, but we do not pursue this issue further here.

Coordination supposes that different programs have equal standings and combines those programs while maximally keeping original information from them. The problem of combining logical theories has been studied by several researchers in different contexts. Baral *et al.* [1] introduce algorithms for combining logic programs by enforcing satisfaction of integrity constraints. For instance, suppose two programs:

$$\begin{aligned} P_1 :\quad & p(x) \leftarrow not\, q(x), \\ & q(b) \leftarrow r(b), \\ & q(a) \leftarrow, \\ P_2 :\quad & r(a) \leftarrow, \end{aligned}$$

together with the integrity constraints:

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

They combine P_1 and P_2 and produce a new program which satisfies IC as follows:

$$\begin{aligned} P_3 :\quad & p(x) \leftarrow not\, q(x),\ x \neq a, \\ & q(b) \leftarrow r(b), \\ & q(a) \vee r(a) \leftarrow . \end{aligned}$$

By contrast, $(P_1 \cup IC) \oplus P_2$ in our framework becomes[5]

$$p(x)\,;\, r(a) \leftarrow not\, q(x),$$

[5] Here IC is included in P_1 as we handle integrity constraints as a part of a program.

$$q(b)\,;\, r(a) \leftarrow r(b),$$
$$q(a)\,;\, r(a) \leftarrow,$$

after eliminating tautologies. Comparing two results, the program P_3 has two answer sets $\{p(b), q(a)\}$ and $\{p(b), r(a)\}$; by contrast, $(P_1 \cup IC) \oplus P_2$ has two answer sets: $\{p(b), q(a)\}$ and $\{r(a)\}$. Thus, the answer sets of P_3 do not coincide with those of the original programs. Indeed, they request that every answer set of a resulting program to be a subset of an answer set of $P_1 \cup P_2$. This is in contrast to our approach where we request the result of coordination to keep (part of) the answer sets of the original programs. Another important difference is that algorithms in [1] are not applicable to unstratified logic programs, while our method is applied to every extended disjunctive program.

The problem of *program composition* has been studied by several researchers (e.g., [4, 6, 12]). It combines different programs into one. The problem is then how to provide the meaning of a program in terms of those components. Brogi *et al.* [4] introduce three meta-level operations for composing normal logic programs: union, intersection, and restriction. The union simply puts two programs together, and the intersection combines two programs by merging pair of rules with unifiable heads. For instance, given two programs:

$$\begin{aligned} P_1:\ & \mathit{likes}(x,y) \leftarrow \mathit{not}\ \mathit{bitter}(y),\\ & \mathit{hates}(x,y) \leftarrow \mathit{sour}(y);\\ P_2:\ & \mathit{likes}(\mathit{Bob},y) \leftarrow \mathit{sour}(y), \end{aligned}$$

the program $P_1 \cap P_2$ consists of the single rule:

$$\mathit{likes}(\mathit{Bob},y) \leftarrow \mathit{not}\ \mathit{bitter}(y),\ \mathit{sour}(y).$$

The restriction allows one to filter out some rules from a program. They employ Fitting's 3-valued fixpoint semantics and show how one can compute the semantics of the composed program in terms of the original programs. In the context of normal open logic programs, Verbaeten *et al.* [12] introduce a variant of the well-founded semantics, and identify conditions for two programs P_1 and P_2 to satisfy the equality $Mod(P_1 \cup P_2) = Mod(P_1) \cap Mod(P_2)$ where $Mod(P)$ is the set of models of P. Etalle and Teusink [6] consider three-valued completion semantics for program composition as the union of normal open programs. Comparing these three studies with ours, both program operations and underlying semantics are different from ours. Moreover, the goal of program composition is to compute the meaning of the whole program in terms of its subprograms; on the other hand, our goal is to construct a program whose answer sets are the union/intersection of the original programs.

Combination of propositional theories has been studied under the names of *merging* [8] or *arbitration* [9]. The goal of these research is to provide a new theory which is consistent and preserves as much information as possible from their sources. Merging is different from coordination presented in this paper. For instance, two theories $P_1 = \{\, p \leftarrow \}$ and $P_2 = \{\, q \leftarrow \}$ are merged into

$P_3 = \{\, p \leftarrow\ ,\ \ q \leftarrow \}$. By contrast, generous coordination of P_1 and P_2 becomes $P_1 \oplus P_2 = \{\, p\,;\, q \leftarrow \}$. Thus, in contrast to generous coordination, merging does not preserve answer sets of the original programs. In merging different beliefs by different agents are mixed together as far as they are consistent, which makes it difficult to distinguish the original beliefs of one agent after merging. This implies the problem that original beliefs of one agent are hard to recover when one of the information sources turns out incorrect. For instance, suppose an agent has the program $P_4 = \{\, p\,;\, q \leftarrow \}$ and new information $P_1 = \{\, p \leftarrow \}$ arrives. If P_4 and P_1 are merged, the result becomes $P_5 = \{\, p \leftarrow \}$. Later, it turns out that the fact p in P_1 does not hold. At this stage, the agent cannot recover the original program P_4 from P_5. By contrast, if generous coordination is done, it becomes $P_4 \oplus P_1 = P_4$ and the original information P_4 is kept.

Ciampolini *et al.* [5] introduce a language for coordinating logic-based agents. They handle two types of coordination: collaboration and competition. Their goal is to solve these different types of queries using abduction, and not to construct a program as a result of coordination. Recently, Meyer *et al.* [10] introduce a logical framework for negotiating agents. They introduce two different modes of negotiation: concession and adaptation. They characterize such negotiation by rational postulates and provide methods for constructing outcomes. Those postulates are not generally applied to nonmonotonic theories, and in this sense coordination considered in this paper is beside the subject of those postulates.

Coordination introduced in this paper is naive in the sense that it just takes the union/intersection of different collections of answer sets. We can develop variants of coordination by introducing strategies that depend on situations. For instance, when there are more than two agents, it is considered to take the *majority* into account as in [8]. Given collections of answer sets by three agents, $\{\, S_1, S_2, S_3 \,\}$, $\{\, S_2, S_4 \,\}$, and $\{\, S_1, S_5 \,\}$, such majority principle allows us to build $\{\, S_1, S_2 \,\}$ as a result of coordination, whose member is supported by more than one agent. Priorities between agents are also considerable. In the above example, if the second agent is most reliable, we can have a choice to take S_4 into account. We can also consider finer grains of compositions such as having $S_1 \cup S_2$ or $S_1 \cap S_2$ as a result of coordination from two answer sets S_1 and S_2 (where $S_1 \cup S_2$ is assumed consistent). Detailed studies on such variants are left to further research.

6 Concluding Remarks

This paper has studied coordination between logical agents. Given multiple agents as logic programs, two different types of coordination have been introduced and their computational methods have been provided. We have verified that the proposed methods realize generous/rigorous coordination between logic programs. Our coordination framework provides a compositional semantics of multiple agents and serves as a declarative basis for accommodation in multi-agent systems. From the viewpoint of answer set programming, the process of computing coordination is considered as a program development under a specification that requests a program reflecting the meanings of two or more programs. This relates to the issue of program composition under the answer set semantics. This paper considered the answer set semantics but a similar framework would be developed under different semantics (though computational methods are likely to be different).

There is still room for improvement in computing generous/rigorous coordination. The operations $\oplus$ and $\otimes$ introduced in this paper require computation of answer sets of original programs, but it is much better if coordination can be constructed by purely syntactic manipulation without computing those answer sets. Further, the operation $\oplus$ produces a disjunctive program even when the original programs are non-disjunctive programs. The resulting disjunctive program is reduced to a non-disjunctive one if it is *head-cycle-free*, but this is not the case in general. At the moment, we do not have solutions for these problems. In future work, we will refine our framework and also investigate other types of coordination and collaboration as well as their characterization in terms of computational logic.

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. S. Brass and J. Dix. Characterizations of the disjunctive stable semantics by partial evaluation. *Journal of Logic Programming* 32(3):207–228, 1997.
4. A. Brogi, S. Contiero, and F. Turini. Composing general logic programs. *Proc. 4th International Conference on Logic Programming and Nonmonotonic Reasoning, Lecture Notes in Artificial Intelligence* 1265, pp. 273–288, Springer, 1997.
5. A. Ciampolini, E. Lamma, P. Mello, F. Toni, and P. Torroni. Cooperation and competition in ALIAS: a logic framework for agents that negotiate. *Annals of Mathematics and Artificial Intelligence* 37(1/2), pp. 65–91, 2003.
6. 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, 1996.
7. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9(3/4):365–385, 1991.
8. 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.
9. P. Liberatore and M. Schaerf. Arbitration (or how to merge knowledge bases). *IEEE Transactions on Knowledge and Data Engineering* 10(1):76–90, 1998.
10. T. Meyer, N. Foo, R. Kwok, and D. Zhang. Logical foundation of negotiation: outcome, concession and adaptation. *Proceedings of the 19th National Conference on Artificial Intelligence*, pp. 293–298, MIT Press, 2004.
11. C. Sakama and H. Seki. Partial deduction in disjunctive logic programming. *Journal of Logic Programming* 32(3):229–245, 1997.
12. 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.