Coordination in Answer Set Programming

CHIAKI SAKAMA
Wakayama University
and
KATSUMI INOUE
National Institute of Informatics

This article studies a semantics of multiple logic programs, and synthesizes a program having such a collective semantics. More precisely, the following two problems are considered: given two logic programs P_1 and P_2, which have the collections 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 answer sets of each program, but permits the introduction of additional answer sets of the other program. By contrast, rigorous coordination forces each program to give up some answer sets, but the result remains within the original answer sets for each program. Coordination provides a program that reflects the meaning of two or more programs. We provide methods for constructing these two types of coordination and address its application to logic-based multi-agent systems.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Logic and constraint programming*; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods—*Representation languages*; I.2.4 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Answer set programming, coordination, multiagent systems

ACM Reference Format:

Sakama, C. and Inoue, K. 2008. Coordination in answer set programming. ACM Trans. Comput. Logic. 9, 2, Article 9 (March 2008), 30 pages. DOI = 10.1145/1342991.1342993 http://doi.acm.org/10.1145/1342991.1342993

This article is a revised and extended version of SAKAMA, C. AND INOUE, K., Coordination between logical agents, In *Proceedings of the 5th International Workshop on Computational Logic in Multi-Agent Systems*, Lecture Notes in Artificial Intelligence, vol. 3487, Springer-Verlag, New York, 2004, 161–177.

Authors' addresses: C. Sakama, Department of Computer and Communication Sciences, Wakayama University, Sakaedani, Wakayama 640 8510, Japan; email: sakama@sys.wakayama-u.ac.jp; K. Inoue, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101 8430, Japan; email: ki@nii.ac.jp.

1. INTRODUCTION

Logic programming provides a formal language for representing knowledge and belief of an agent. The declarative semantics of a program is given by a set of canonical models which represent belief sets of an agent. Our primary interest in this article is: what is a suitable semantics of multiple programs, and how to synthesize a program having such a collective semantics. Those problems become especially important when an individual agent has a program in multi-agent environments. In multi-agent systems, different agents may have different sets of beliefs, and agents accommodate themselves to reach acceptable agreements. We call the process of forming such agreements between agents *coordination*. In multi-agent environments, individual agents are supposed to have incomplete information and solve problems cooperatively. In logic programming, incomplete information is represented in the framework of *nonmonotonic logic programming* or *answer set programming* [Lifschitz 2002]. We then use answer set programming for representing and specifying agents.

In answer set programming, a problem is described by an *extended disjunctive program* [Gelfond and Lifschitz 1991] and solutions are computed by *answer sets* of the program. Answer sets represent sets of literals corresponding to beliefs which can be built by a rational reasoner on the basis of a program [Baral and Gelfond 1994]. Suppose an agent that has a knowledge base represented by an extended disjunctive program. 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 article, we consider two different solutions: one is a program Q that has three answer sets S_1, S_2, and S_3; the other is a program R that 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. 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 coordinations between agents.

The problem is then how to build a program that realizes such coordination. Formally, the problems considered in this article 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 investigate their properties. Coordination among more than two programs is also obtained using operational properties of each coordination.

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. We show that generous coordination merges consequences of *credulous reasoning* in two programs, while restricts consequences of *skeptical reasoning* to those that are common between two programs. By contrast, rigorous coordination reduces credulous consequences, but increases skeptical consequences in general. The problem of coordinating logic programs relates to the issue of *program composition* or *merging*. However, our goal is different from program composition or merging. Program composition aims at obtaining the meaning of a program by its component [Brogi 2004], while our purpose is to coordinate meanings of different programs. Program merging has outcomes different from coordination in general. In the above example, generous coordination Q of P_1 and P_2 has three answer sets $\{f_1\}$, $\{f_2\}$, and $\{f_3\}$, and rigorous coordination R of P_1 and P_2 has the single answer set $\{f_2\}$. However, merging $P_1 \cup P_2$ produces two answer sets $\{f_1, f_3\}$ and $\{f_2\}$. We will argue the issue in more detail in Section 7.

The rest of this article is organized as follows. Section 2 presents definitions and terminologies used in this article. Section 3 introduces a framework of coordination between logic programs. Section 4 provides methods for computing coordination, and Section 5 addresses their properties. Section 6 exhibits various extensions in multi-agent coordination. Section 7 discusses related issues and Section 8 summarizes this article.

2. PRELIMINARIES

We suppose an agent that has a knowledge base represented with answer set programming. An agent is then identified with its logic program and we use those terms interchangeably throughout the article.

A *program* considered in this article is an *extended disjunctive program* (EDP) which is a set of *rules* of the form:

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

where each L_i is a positive/negative literal, namely, A or $\neg A$ for an atom A. *not* is *negation as failure* (NAF), and *not* L is called an *NAF-literal*. A negative literal represents negative information explicitly as in classical logic, while an NAF-literal represents negative information implicitly under the closed world assumption. 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, $\mathit{head}(r)$, $\mathit{body}^+(r)$, $\mathit{body}^-(r)$, and $\mathit{not\_body}^-(r)$ denote the sets of (NAF-) literals $\{L_1, \ldots, L_l\}$, $\{L_{l+1}, \ldots, L_m\}$, $\{L_{m+1}, \ldots, L_n\}$, and $\{\mathit{not}\, L_{m+1}, \ldots, \mathit{not}\, L_n\}$, 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 $\mathit{head}(r) \leftarrow \mathit{body}^+(r), \mathit{not\_body}^-(r)$ or $\mathit{head}(r) \leftarrow \mathit{body}(r)$ where $\mathit{body}(r) = \mathit{body}^+(r) \cup \mathit{not\_body}^-(r)$. A rule r is *disjunctive* if $\mathit{head}(r)$ contains more than one literal. A rule r is an *integrity constraint* if $\mathit{head}(r) = \emptyset$; and r is a *fact* if $\mathit{body}(r) = \emptyset$. A program P is *NAF-free* if $\mathit{body}^-(r) = \emptyset$ for every rule r in P. A program P with variables is semantically identified with its ground instantiation, that is, the set of ground rules obtained from P by substituting variables in P by elements of its Herbrand universe in every possible way. Every program containing variables is then considered as a shorthand of its ground instantiation, and this paper handles ground programs unless stated otherwise.

The semantics of EDPs is given by the *answer set semantics* [Gelfond and Lifschitz 1991]. Let *Lit* be the set of all ground literals in the language of a program. A set $S \subseteq \mathit{Lit}$ *satisfies* a ground rule r if $\mathit{body}^+(r) \subseteq S$ and $\mathit{body}^-(r) \cap S = \emptyset$ imply $\mathit{head}(r) \cap S \neq \emptyset$. In particular, S satisfies a ground integrity constraint r with $\mathit{head}(r) = \emptyset$ if either $\mathit{body}^+(r) \not\subseteq S$ or $\mathit{body}^-(r) \cap S \neq \emptyset$. S satisfies a ground program P if S satisfies every rule in P. When $\mathit{body}^+(r) \subseteq S$ (respectively, $\mathit{head}(r) \cap S \neq \emptyset$), it is also written as $S \models \mathit{body}^+(r)$ (respectively, $S \models \mathit{head}(r)$).

Let P be an NAF-free EDP. Then, a set $S \subseteq \mathit{Lit}$ is an *answer set* of P if S is a minimal set such that

(1) S satisfies every rule from the ground instantiation of P,
(2) $S = \mathit{Lit}$ if S contains a pair of complementary literals L and $\neg L$.

Next, let P be any EDP and $S \subseteq \mathit{Lit}$. For every rule r in the ground instantiation of P, the rule $r^S : \ \mathit{head}(r) \leftarrow \mathit{body}^+(r)$ is included in the *reduct* P^S if $\mathit{body}^-(r) \cap S = \emptyset$. 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 $\mathcal{AS}(P)$. 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 the single answer set *Lit* or no answer set. An inconsistent program with the answer set *Lit* is called *contradictory*, while an inconsistent program with no answer set is called *incoherent*.

A literal L is a consequence of *skeptical reasoning* (respectively, *credulous reasoning*) in a program P if L is included in every (respectively, some) answer set of P. The set of all (literal) consequences under skeptical (respectively, credulous) reasoning in P is written as $skp(P)$ (respectively, $crd(P)$). Note that, by the definition, $skp(P) = Lit$ and $crd(P) = \emptyset$ if P is incoherent; and $skp(P) = crd(P) = Lit$ if P is contradictory. Clearly, $skp(P) \subseteq crd(P)$ for any consistent program 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. Consider 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. Here, *Lit* is the set of literals in the language of $P_1 \cup P_2$, and $\overline{L}$ is a newly introduced atom associated uniquely with each L. Let $\mathcal{AS}(P)\,|_{Lit} = \{S \cap Lit \mid S \in \mathcal{AS}(P)\}$. Then, the following property holds.

PROPOSITION 2.1. *For two programs P_1 and P_2, put $P_1' = P_1 \cup N$ and $P_2' = P_2 \cup N$ where $N = \{\overline{L} \leftarrow not\, L \mid L \in Lit\}$. Then, P_1' and P_2' are AS-combinable and $\mathcal{AS}(P_i) = \mathcal{AS}(P_i')\,|_{Lit}$ $(i = 1, 2)$.*[1]

PROOF. When P_1 and P_2 are not AS-combinable, there are $S \in \mathcal{AS}(P_1)$ and $T \in \mathcal{AS}(P_2)$ such that $S \subseteq T$. Correspondingly, there are $S' \in \mathcal{AS}(P_1')$ and $T' \in \mathcal{AS}(P_2')$ such that $S' = S \cup \{\overline{L} \mid L \notin S\}$ and $T' = T \cup \{\overline{L} \mid L \notin T\}$. By $S \subseteq T$, there is a literal $M \in T \backslash S$, which implies $M \in T' \backslash S'$ and $\overline{M} \in S' \backslash T'$. Thus, $T' \not\subseteq S'$ and $S' \not\subseteq T'$. Hence, P_1' and P_2' are AS-combinable. By $S' \cap Lit = S$ and $T' \cap Lit = T$, $\mathcal{AS}(P_i) = \mathcal{AS}(P_i')\,|_{Lit}$ $(i = 1, 2)$. Else when P_1 and P_2 are AS-combinable, $S \not\subseteq T$ nor $T \not\subseteq S$ for any $S \in \mathcal{AS}(P_1)$ and $T \in \mathcal{AS}(P_2)$. This implies $S' \not\subseteq T'$ and $T' \not\subseteq S'$. □

[1]The second part of the proposition, that is, the 1-1 correspondence between the answer sets of P_i and the answer sets of P_i', is also independently derived by Erdoğan and Lifschitz [2004, Proposition 3].

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

$$N: \quad \overline{p} \leftarrow not\ p, \qquad \overline{q} \leftarrow not\ q\ .$$

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

In this section, we introduce two different types of coordination in answer set programming. Throughout the paper, different programs are assumed to have the same underlying language. This implies that every program has the same set *Lit* of all ground literals in the language.

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. $\mathcal{AS}(Q)$ (respectively, $\mathcal{AS}(R)$) is called the *result* of generous (respectively, rigorous) coordination.

The above program Q or R is also called a (generous or rigorous) *coordinated program* (of P_1 and P_2).[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. So 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.

By Definition 3.1, generous coordination produces no answer set $\mathcal{AS}(Q) = \emptyset$ when $\mathcal{AS}(P_1) = \mathcal{AS}(P_2) = \emptyset$. Similarly, rigorous coordination has no answer set $\mathcal{AS}(R) = \emptyset$ when either $\mathcal{AS}(P_1) = \emptyset$ or $\mathcal{AS}(P_2) = \emptyset$. In each case, the coordination result is trivial and uninteresting. Coordination is considered successful if a coordinated program has an answer set.

Definition 3.2. Let Q be a generous coordinated program, and R a rigorous coordinated program. We say that generous (respectively, rigorous) coordination *succeeds* if $\mathcal{AS}(Q) \neq \emptyset$ (respectively, $\mathcal{AS}(R) \neq \emptyset$); otherwise, it *fails*.

Generous coordination succeeds when at least one program has an answer set. By contrast, rigorous coordination succeeds only when two programs have a common answer set. Even when generous/rigorous coordination succeeds, the result becomes inconsistent if two programs are both contradictory, namely,

[2]We will often omit "of P_1 and P_2" when it is clear from the context.

$\mathcal{AS}(Q) = \mathcal{AS}(R) = \{Lit\}$ if $\mathcal{AS}(P_1) = \mathcal{AS}(P_2) = \{Lit\}$. A successful coordination is consistent when original programs are consistent.

PROPOSITION 3.1. *Given two consistent programs, generous/rigorous coordination succeeds iff the result of coordination is consistent.*

PROOF. Let Q (respectively, R) be a generous (respectively, rigorous) coordinated program of two consistent programs. Then, generous (respectively, rigorous) coordination succeeds iff $\mathcal{AS}(Q) \neq \emptyset$ (respectively, $\mathcal{AS}(R) \neq \emptyset$) (Definition 3.2) iff there is a consistent answer set $S \in \mathcal{AS}(Q)$ (respectively, $T \in \mathcal{AS}(R)$). Hence, the result follows. □

Our primary interest is successful and consistent coordination, so that hereafter every program is assumed to be consistent unless stated otherwise. Note that generous coordination may produce a collection of answer sets which contradict with one another. But this does not cause any problem, since even a single program may have a collection of conflicting answer sets. A collection of answer sets represents (conflicting) alternative belief sets of each agent.

The next proposition presents relations between consequences of credulous/skeptical reasoning in individual programs and those in coordinated programs.

PROPOSITION 3.2. *Let P_1 and P_2 be two consistent programs.*

(1) *If Q is a generous coordinated program,*
   (a) $crd(Q) = crd(P_1) \cup crd(P_2)$;
   (b) $skp(Q) = skp(P_1) \cap skp(P_2)$;
   (c) $crd(Q) \supseteq crd(P_i)$ *for* $i = 1, 2$;
   (d) $skp(Q) \subseteq skp(P_i)$ *for* $i = 1, 2$.

(2) *If R is a rigorous coordinated program,*
   (a) $crd(R) \subseteq crd(P_1) \cup crd(P_2)$;
   (b) $skp(R) \supseteq skp(P_1) \cup skp(P_2)$;
   (c) $crd(R) \subseteq crd(P_i)$ *for* $i = 1, 2$;
   (d) $skp(R) \supseteq skp(P_i)$ *for* $i = 1, 2$.

PROOF

(1) (a) For any $L \in Lit$, $L \in crd(Q)$
iff $\exists S \in \mathcal{AS}(P_1) \cup \mathcal{AS}(P_2)$ such that $L \in S$
iff $\exists S_1 \in \mathcal{AS}(P_1)$ such that $L \in S_1$, or $\exists S_2 \in \mathcal{AS}(P_2)$ such that $L \in S_2$
iff $L \in crd(P_1) \cup crd(P_2)$.

(b) For any $L \in Lit$, $L \in skp(Q)$
iff $\forall S \in \mathcal{AS}(P_1) \cup \mathcal{AS}(P_2),\ L \in S$
iff $\forall S_1 \in \mathcal{AS}(P_1),\ L \in S_1$; and $\forall S_2 \in \mathcal{AS}(P_2),\ L \in S_2$
iff $L \in skp(P_1) \cap skp(P_2)$.

The results of (c) and (d) hold by (a) and (b), respectively.

(2) (a) For any $L \in Lit$, $L \in crd(R)$
iff $\exists S \in \mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)$ such that $L \in S$
only if $\exists S_1 \in \mathcal{AS}(P_1)$ such that $L \in S_1$, and $\exists S_2 \in \mathcal{AS}(P_2)$ such that $L \in S_2$
iff $L \in crd(P_1) \cup crd(P_2)$.

(b) For any $L \in Lit$, $L \in skp(R)$
iff $\forall S \in \mathcal{AS}(P_1) \cap \mathcal{AS}(P_2),\ L \in S$
if $\forall S_1 \in \mathcal{AS}(P_1),\ L \in S_1$; or $\forall S_2 \in \mathcal{AS}(P_2),\ L \in S_2$
iff $L \in skp(P_1) \cup skp(P_2)$.

The results of (c) and (d) hold by (a) and (b), respectively. □

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 $crd(P_1) = \{a, b, c, d\}$, $skp(P_1) = \{b, c\}$, $crd(P_2) = \{b, c, d, e\}$, and $skp(P_2) = \{c\}$. A generous coordinated program Q has the answer sets $\mathcal{AS}(Q) = \{\{a, b, c\}, \{b, c, d\}, \{c, e\}\}$ where $crd(Q) = \{a, b, c, d, e\}$ and $skp(Q) = \{c\}$. A rigorous coordinated program R has the answer sets $\mathcal{AS}(R) = \{\{b, c, d\}\}$ where $crd(R) = skp(R) = \{b, c, d\}$. 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 generous coordinated program, and R a rigorous coordinated program. When $\mathcal{AS}(Q) = \mathcal{AS}(P_1)$ (respectively, $\mathcal{AS}(R) = \mathcal{AS}(P_1)$), P_1 *dominates* P_2 under generous (respectively, rigorous) coordination.

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

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.*

PROOF. When $\mathcal{AS}(P_1) \subseteq \mathcal{AS}(P_2)$, $\mathcal{AS}(P_1) \cup \mathcal{AS}(P_2) = \mathcal{AS}(P_2)$ and $\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2) = \mathcal{AS}(P_1)$. So P_2 dominates P_1 under generous coordination, and P_1 dominates P_2 under rigorous coordination. □

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 when $\mathcal{AS}(P_1) \not\subseteq \mathcal{AS}(P_2)$ 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 coordinated programs.

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,

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

where $head(r_1)\,;\, head(r_2)$ is the disjunction of $head(r_1)$ and $head(r_2)$, and

$$\begin{aligned} body_*(r_1) = body(r_1) \backslash \{\, not\, L \mid L \in T \text{ for some } T \in \mathcal{AS}(P_2) \text{ and} \\ L \notin S \text{ for any } S \in \mathcal{AS}(P_1)\,\}, \\ body_*(r_2) = body(r_2) \backslash \{\, not\, L \mid L \in S \text{ for some } S \in \mathcal{AS}(P_1) \text{ and} \\ L \notin T \text{ for any } T \in \mathcal{AS}(P_2)\,\}. \end{aligned}$$

In $P_1 \oplus P_2$, each rule is obtained by disjunctively (respectively, conjunctively) combining heads (respectively, bodies) of rules from P_1 and P_2. In combining bodies, every NAF-literal $not\, L$ such that $L \in T$ for some $T \in \mathcal{AS}(P_2)$ and $L \notin S$ for any $S \in \mathcal{AS}(P_1)$ is dropped from $body_*(r_1)$, because the existence of such literals may prevent the derivation of some literal in $head(r_2)$ after combination.

Example 4.1. Consider two programs:

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

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 first rule of $P_1 \oplus P_2$ because of the existence of $\{p\}$ in $\mathcal{AS}(P_1)$.

By the definition, $P_1 \oplus P_2$ requires computation of all answer sets of P_1 and P_2, which generally takes exponential time in the size of programs. However, if $\mathcal{AS}(P_1)$ and $\mathcal{AS}(P_2)$ are computed by an answer set solver and given as an input, the additional cost for computing $P_1 \oplus P_2$ is $O(|P_1| \times |P_2| \times |\mathcal{AS}(P_1)| \times |\mathcal{AS}(P_2)|)$, where $|P|$ represents the number of rules in P and $|\mathcal{AS}(P)|$ represents the number of answer sets in P. Note that if both P_1 and P_2 are NAF-free, $body_*(r_i) = body(r_i)$. In this case, $P_1 \oplus P_2$ is obtained in time $O(|P_1| \times |P_2|)$ without computing answer sets of P_1 and P_2.

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 [Brass and Dix 1997].

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 consistent 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. Let S be an answer set of P_1. Then, S satisfies every rule $head(r_1) \leftarrow body(r_1)$ in P_1, thereby satisfies every 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 set $T \subset S$ which satisfies every rule in $P_1 \oplus P_2$. Since S is an answer set of P_1, T is not an answer set of P_1. So, 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, namely, $T' \not\subset S$. 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, let S be an answer set of $P_1 \oplus P_2$. Then, S satisfies every 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 $S \models head(r_1); head(r_2)$. (i) If $S \not\models head(r_1)$ for some rule $r_1 \in P_1$, $S \models head(r_2)$ for any $r_2 \in P_2$. In this case, $S \models body(r_2)$ implies $S \models head(r_2)$ for any $r_2 \in P_2$, so that S satisfies every rule in P_2. (ii) Else if $S \not\models head(r_2)$ for some rule $r_2 \in P_2$, $S \models head(r_1)$ for any $r_1 \in P_1$. In this case, $S \models body(r_1)$ implies $S \models head(r_1)$ for any $r_1 \in P_1$, so that S satisfies every rule in P_1. (iii) Else if $S \models head(r_1)$ for any $r_1 \in P_1$ and $S \models head(r_2)$ for any $r_2 \in P_2$, 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 done when S satisfies P_2. □

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

PROOF. Let $S \in \mathcal{AS}(P_1)$. 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.) To prove $S \in \mathcal{AS}(P_1 \oplus P_2)$, we compare $P_1^S \oplus P_2^T$ and $(P_1 \oplus P_2)^S$. First, 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 $body^-(r_1) \cap S = body^-(r_2) \cap T = \emptyset$. Next, 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 $body_*^-(r_1) \cap S = body_*^-(r_2) \cap S = \emptyset$. By $body_*^-(r_1) \subseteq body^-(r_1)$, it holds that $body_*^-(r_1) \cap S \subseteq body^-(r_1) \cap S$. When $body_*(r_2)$ contains an NAF-literal *not* L with $L \in S$, there is $T \in \mathcal{AS}(P_2)$ such that $L \in T$ (Definition 4.1). Thus, every literal included in $body_*^-(r_2) \cap S$ is also included in T. Then, $body_*^-(r_2) \cap S = body_*^-(r_2) \cap S \cap T$ holds. Since $body_*^-(r_2) \cap S \cap T \subseteq body_*^-(r_2) \cap T$, it holds that $body_*^-(r_2) \cap S \subseteq body_*^-(r_2) \cap T \subseteq body^-(r_2) \cap T$. By $body_*^-(r_1) \cap S \subseteq body^-(r_1) \cap S$ and $body_*^-(r_2) \cap S \subseteq body^-(r_2) \cap T$, it holds that $P_1^S \oplus P_2^T \subseteq (P_1 \oplus P_2)^S$. Suppose any rule $head(r_1); head(r_2) \leftarrow body^+(r_1), body^+(r_2)$ in $(P_1 \oplus P_2)^S \backslash (P_1^S \oplus P_2^T)$. Since S satisfies any rule r_1 in P_1, (i) $S \cap head(r_1) \neq \emptyset$ or (ii) $body^+(r_1) \not\subseteq S$ or (iii) $body^-(r_1) \cap S \neq \emptyset$. In case of (i), $S \cap head(r_1) \neq \emptyset$ implies $S \models head(r_1); head(r_2)$. In case of (ii), $body^+(r_1) \not\subseteq S$ implies $S \not\models body^+(r_1), body^+(r_2)$. In case of (iii), $body^-(r_1) \cap S \neq \emptyset$ implies $body_*^-(r_1) \cap S = (body^-(r_1) \backslash \{ L \mid L \in T \text{ for some } T \in \mathcal{AS}(P_2) \text{ and } L \notin S \text{ for any } S \in \mathcal{AS}(P_1) \}) \cap S \neq \emptyset$. Thus, in cases of (i) and (ii), a rule $head(r_1); head(r_2) \leftarrow body^+(r_1), body^+(r_2)$ is included in $(P_1 \oplus P_2)^S \backslash (P_1^S \oplus P_2^T)$, and S satisfies the rule. In case of (iii), the rule is not included in the reduct $(P_1 \oplus P_2)^S$. Hence, the answer set S of $P_1^S \oplus P_2^T$ satisfies every rule in $(P_1 \oplus P_2)^S \backslash (P_1^S \oplus P_2^T)$. 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, let $U \in \mathcal{AS}(P_1 \oplus P_2)$. Then, U is an answer set of $(P_1 \oplus P_2)^U$. Put $P_i' = \{ head(r_i) \leftarrow body_*^+(r_i) \mid head(r_1); head(r_2) \leftarrow body_*^+(r_1), body_*^+(r_2) \in (P_1 \oplus P_2)^U \}$ for $i = 1, 2$. Then, $(P_1 \oplus P_2)^U = P_1' \oplus P_2'$. So U is an answer set of $P_1' \oplus P_2'$, and that U is an answer set of either P_1' or P_2' (by Lemma 4.1). Suppose

$U \in \mathcal{AS}(P_1')$. Let $P_{1*} = \{ head(r_1) \leftarrow body_*(r_1) \mid r_1 \in P_1 \}$ where $body_*^-(r_1) = body^-(r_1) \backslash V$ with $V = \{ L \mid L \in T$ for some $T \in \mathcal{AS}(P_2)$ and $L \notin S$ for any $S \in \mathcal{AS}(P_1) \}$. Then, $P_1' = P_{1*}^U$. Thus, U is an answer of P_1' iff U is an answer set of P_{1*}^U. Since any literal in V is included in no answer set of P_1, removing V from $body^-(r_1)$ does not affect the construction of answer sets of P_1. Hence, U is an answer set of P_{1*} iff U is an answer set of P_1. Similarly, $U \in \mathcal{AS}(P_2')$ implies $U \in \mathcal{AS}(P_2)$. Therefore, U is an answer set of either P_1 or P_2, and the result holds. □

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

Note that in Theorem 4.2, P_1 and P_2 must be AS-combinable because every set in $\mathcal{AS}(P_1 \oplus P_2)$ is minimal under set inclusion. However, the result can be applied to non-AS-combinable programs P_1 and P_2 by making them AS-combinable using the program transformation presented in Section 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 such that $\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2) \neq \emptyset$,

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

where

$$R(P, S) = \{ head(r^*) \leftarrow body(r),\ not\_body^-(r^*) \mid r \in P \text{ and } r^S \in P^S \}$$

with

$$head(r^*) = head(r) \cap S,$$
$$not\_body^-(r^*) = \{ not\ L \mid L \in head(r) \backslash S \}.$$

When $\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2) = \emptyset$, define $P_1 \otimes P_2 = \{ p \leftarrow not\ p \}$ for any atom p.

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 $R(P, S)$ every literal in $head(r)$ which does not contribute to the construction of the answer set S is shifted to the body as NAF-literals. Note that a set T of literals satisfies $r \in P$ iff T satisfies $r' \in R(P, S)$ for any S. This explains the need of introducing $not\_body^-(r^*)$ to the body of every rule in $R(P, S)$. When $\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2) = \emptyset$, $P_1 \otimes P_2$ has no answer set by the definition. $P_1 \otimes P_2$ may contain redundant rules, which are eliminated using program transformations given in Section 4.1. For notational convenience, we often abuse notation and write $head(r^*) \leftarrow body(r)$, $not\_body^-(r^*)$ in $R(P, S)$ as $head(r) \cap S \leftarrow body(r)$, $not\ (head(r) \backslash S)$.

Example 4.4. Consider two programs:

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

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

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

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

As the case of generous coordination, $P_1 \otimes P_2$ generally requires exponential computation of all answer sets of P_1 and P_2. When $\mathcal{AS}(P_1)$ and $\mathcal{AS}(P_2)$ are given as an input, however, $P_1 \otimes P_2$ is computed in time $O((|P_1| + |P_2|) \times |\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)|)$ where $|\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)|$ represents the number of answer sets in $\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)$.

Now we show that $P_1 \otimes P_2$ realizes rigorous coordination of P_1 and P_2.

LEMMA 4.3. *Let P be a consistent 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 $body^+(r) \subseteq S$ implies $head(r) \cap S \neq \emptyset$ for any rule $head(r) \leftarrow body^+(r)$ in P^S $(*)$. By the definition of $R(P, S)$, the rule $head(r) \leftarrow body^+(r)$ is in P^S iff the corresponding rule $head(r) \cap S \leftarrow body^+(r)$ is in $R(P, S)^S$ (because $body^-(r) \cap S = \emptyset$ and $(head(r) \backslash S) \cap S = \emptyset$). Hence, the statement $(*)$ holds iff S is a minimal set such that $body^+(r) \subseteq S$ implies $head(r) \cap S \neq \emptyset$ for any rule $head(r) \cap S \leftarrow body^+(r)$ in $R(P, S)^S$ iff S is a minimal set which satisfies every rule $head(r) \cap S \leftarrow body^+(r)$ 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 consistent programs. Then, $\mathcal{AS}(P_1 \otimes P_2) = \mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)$.*

PROOF. Let $S \in \mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)$. Then, S satisfies every rule $head(r) \leftarrow body(r)$ in P_1 and P_2, so that S satisfies corresponding each rule $head(r) \cap T \leftarrow body(r)$, $not\,(head(r) \backslash T)$ 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 set $U \subset S$ which satisfies every rule in $(P_1 \otimes P_2)^S$. In this case, U satisfies $R(P_1, S)^S$. By Lemma 4.3, however, $S \in \mathcal{AS}(P_1)$ implies that S is a minimal set which satisfies $R(P_1, S)^S$. Contradiction. Hence, S is an answer set of $P_1 \otimes P_2$.

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

5. PROPERTIES

5.1 Operational Properties

The operations $\oplus$ and $\otimes$ have the following properties.

PROPOSITION 5.1. *Let P_1, P_2, and P_3 be programs. Then,*

(1) $P_1 \oplus P_2 = P_2 \oplus P_1$ *and* $(P_1 \oplus P_2) \oplus P_3 = P_1 \oplus (P_2 \oplus P_3)$*;*
(2) $P_1 \otimes P_2 = P_2 \otimes P_1$ *and* $(P_1 \otimes P_2) \otimes P_3 = P_1 \otimes (P_2 \otimes P_3)$.

PROOF. The commutative law is straightforward, so we show the associative law.

(1) $(P_1 \oplus P_2) \oplus P_3$ contains rules of the form:

$$head(r_1); head(r_2); head(r_3) \leftarrow body_{**}(r_1), body_{**}(r_2), body_{*}(r_3).$$

Here, $body_{**}(r_1) = (body(r_1) \backslash D_1) \backslash D_2$ where $D_1 = \{not\, L \mid L \in T$ for some $T \in \mathcal{AS}(P_2)$ and $L \notin S$ for any $S \in \mathcal{AS}(P_1)\}$ and $D_2 = \{not\, M \mid M \in U$ for some $U \in \mathcal{AS}(P_3)$ and $M \notin V$ for any $V \in \mathcal{AS}(P_1) \cup \mathcal{AS}(P_2)\}$. Now, it holds that $D_2 = D_3 \backslash D_4$ where $D_3 = \{not\, M \mid M \in U$ for some $U \in \mathcal{AS}(P_3)$ and $M \notin V$ for any $V \in \mathcal{AS}(P_1)\}$ and $D_4 = \{not\, M \mid M \in U$ for some $U \in \mathcal{AS}(P_3)$ and $M \notin V$ for any $V \in \mathcal{AS}(P_1)$ and $M \in$ for some $W \in \mathcal{AS}(P_2)\}$. Since $D_4 \subseteq D_1$, $body_{**}(r_1) = (body(r_1) \backslash D_1) \backslash D_2 = (body(r_1) \backslash D_1) \backslash (D_3 \backslash D_4) = body(r_1) \backslash (D_1 \cup D_3)$. Thus, $body_{**}(r_1) = body(r_1) \backslash \{not\, L \mid L \in T$ for some $T \in \mathcal{AS}(P_2) \cup \mathcal{AS}(P_3)$ and $L \notin S$ for any $S \in \mathcal{AS}(P_1)\}$. Similarly, $body_{**}(r_2) = body(r_2) \backslash \{not\, L \mid L \in T$ for some $T \in \mathcal{AS}(P_1) \cup \mathcal{AS}(P_3)$ and $L \notin S$ for any $S \in \mathcal{AS}(P_2)\}$, and $body_{*}(r_3) = body(r_3) \backslash \{not\, L \mid L \in$ for some $T \in \mathcal{AS}(P_1) \cup \mathcal{AS}(P_2)$ and $L \notin S$ for any $S \in \mathcal{AS}(P_3)\}$. It is verified that $P_1 \oplus (P_2 \oplus P_3)$ consists of the same rules by replacing r_1 with r_2, r_2 with r_3, and r_3 with r_1 in the above proof.

(2) When $\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2) \cap \mathcal{AS}(P_3) = \emptyset$, the result holds by putting $(P_1 \otimes P_2) \otimes P_3 = P_1 \otimes (P_2 \otimes P_3) = \{\, p \leftarrow not\, p \,\}$ for some atom p. Let $\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2) \cap \mathcal{AS}(P_3) \neq \emptyset$. Suppose a rule $head(r_i) \cap S \leftarrow body(r_i)$, $not\ (head(r_i) \backslash S)$ in $P_1 \otimes P_2$ where $r_i \in P_i$ $(i = 1, 2)$ and $S \in \mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)$. Then, for every $T \in \mathcal{AS}(P_1) \cap \mathcal{AS}(P_2) \cap \mathcal{AS}(P_3)$, $(P_1 \otimes P_2) \otimes P_3$ consists of rules

$$u_i = (head(r_i) \cap S) \cap T \leftarrow body(r_i),\ not\ (head(r_i) \backslash S),\ not\ ((head(r_i) \cap S) \backslash T)$$

$(i = 1, 2)$ where $u_i^T \in (P_1 \otimes P_2)^T$; and rules

$$head(r_3) \cap T \leftarrow body(r_3),\ not\ (head(r_3) \backslash T)$$

where $r_3 \in P_3$ and $r_3^T \in P_3^T$. For every rule u_i, if $T = S$, u_i is equivalent to $head(r_i) \cap T \leftarrow body(r_i)$, *not* $(head(r_i) \backslash T)$. Else, if $T \neq S$, there are literals $L, M \in Lit$ such that $L \in S \backslash T$ and $M \in T \backslash S$. (a) When $M \in head(r_i)$, $M \in head(r_i) \backslash S$. Them, u_i^T is not included in $(P_1 \otimes P_2)^T$. (b) Else when $M \notin head(r_i)$ and $L \in head(r_i)$, $head(r_i) \cap S \cap T = head(r_i) \cap T$. As $head(r_i) \backslash S = (head(r_i) \backslash S) \backslash T$, $(head(r_i) \backslash S) \cup ((head(r_i) \cap S) \backslash T) = head(r_i) \backslash T$. Hence, u_i is equivalent to $head(r_i) \cap T \leftarrow body(r_i)$, *not* $(head(r_i) \backslash T)$. (c) Else when $M \notin head(r_i)$ and $L \notin head(r_i)$, $head(r_i) \cap S \cap T = head(r_i) \cap T$ and $(head(r_i) \backslash S) \cup ((head(r_i) \cap S) \backslash T) = head(r_i) \backslash T$. Then, u_i is equivalent to $head(r_i) \cap T \leftarrow body(r_i)$, *not* $(head(r_i) \backslash T)$. Hence, $(P_1 \otimes P_2) \otimes P_3$ consists of rules:

$$head(r_i) \cap T \leftarrow body(r_i),\ not\ (head(r_i) \backslash T)\ (i = 1, 2, 3)$$

for $T \in \mathcal{AS}(P_1) \cap \mathcal{AS}(P_2) \cap \mathcal{AS}(P_2)$. Similarly, it is shown that $P_1 \otimes (P_2 \otimes P_3)$ consists of the same set of rules. Therefore, the result holds. □

Thus, $\oplus$ and $\otimes$ are both commutative and associative. Moreover, $\oplus$ is also *idempotent*, $P \oplus P = P$ if NONMIN and DUPL are applied to $P \oplus P$ and P. By contrast, $\otimes$ is not idempotent, but the relation $\mathcal{AS}(P \otimes P) = \mathcal{AS}(P)$ holds.

By Proposition 5.1, when generous/rigorous coordination is done among more than two agents, the order of computing coordination does not affect the final outcome. Two types of coordination are mixed among agents. In this case, the absorption laws and the distribution laws do not hold in general, that is:

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

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

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

Given two programs P_1 and P_2, suppose programs P_1' and P_2' such that $\mathcal{AS}(P_1) = \mathcal{AS}(P_1')$ and $\mathcal{AS}(P_2) = \mathcal{AS}(P_2')$. Then, by Theorems 4.2 and 4.4, it holds that $\mathcal{AS}(P_1 \oplus P_2) = \mathcal{AS}(P_1' \oplus P_2')$ and $\mathcal{AS}(P_1 \otimes P_2) = \mathcal{AS}(P_1' \otimes P_2')$. This means that when the original programs are transformed to semantically equivalent programs, such transformations do not change the results of generous/rigorous coordination. In this case, however, coordinated programs $P_1 \oplus P_2$ and $P_1' \oplus P_2'$ (or $P_1 \otimes P_2$ and $P_1' \otimes P_2'$) have different syntax in general.

5.2 Syntactic Properties

In Section 4, we provided methods for constructing programs that reflect the meaning of generous/rigorous coordination. One may wonder the need of creating coordinated programs ($P_1 \oplus P_2$ or $P_1 \otimes P_2$), given the results ($\mathcal{AS}(P_1) \cup \mathcal{AS}(P_2)$ or $\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)$) of coordination. To explain the reason, consider the

programs:

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

where $\mathcal{AS}(P_1) = \{\{\mathit{sweet}, \mathit{strawberry}\}\}$ and $\mathcal{AS}(P_2) = \{\{\mathit{red}, \mathit{strawberry}\}\}$. The result of rigorous coordination is $\mathcal{AS}(P_1) \cap \mathcal{AS}(P_2) = \emptyset$, and the result of generous coordination is $\mathcal{AS}(P_1) \cup \mathcal{AS}(P_2) = \{\{\mathit{sweet}, \mathit{strawberry}\}, \{\mathit{red}, \mathit{strawberry}\}\}$. As rigorous coordination fails, we consider generous coordination. A generous coordinated program $P_1 \oplus P_2$ becomes

$$\begin{array}{l} \mathit{sweet}; \mathit{red} \leftarrow \mathit{strawberry}, \\ \mathit{strawberry} \leftarrow, \end{array}$$

after eliminating duplicated literals and redundant rules. The benefit of having the coordinated program $P_1 \oplus P_2$, in addition to the result $\mathcal{AS}(P_1) \cup \mathcal{AS}(P_2)$, is explained as follows. A simple result of coordination, $\{\mathit{sweet}, \mathit{strawberry}\}$ or $\{\mathit{red}, \mathit{strawberry}\}$, does not bring information on which the coordination is ground. The agent P_1 does not know why P_2 has the belief set $\{\mathit{red}, \mathit{strawberry}\}$. When an agent accepts different belief sets of another agent as a result of coordination, information on which the belief sets are ground would be helpful to understand the mental state of another agent. The program $P_1 \oplus P_2$ represents background knowledge which accounts for the result of coordination. Generally, a program—a collection of rules, contains more information than its answer set—a collection of literals. In multi-agent environments, a coordinated program serves as a social knowledge base consented among agents. Agents would act and make decisions in a society based on a coordinated program. This explains the reason of constructing a coordinated program, even after obtaining the result of coordination.

When a set of answer sets is given, however, 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: $R_1 \wedge \cdots \wedge R_n$. Since these are logically equivalent transformations, the set of facts $\{R_1, \ldots, R_n\}$ has the answer sets $\{S_1, \ldots, S_m\}$. This technique is also used for computing coordination between programs. For instance, to get a generous coordinated program which has the answer sets $\mathcal{AS}(P_1) \cup \mathcal{AS}(P_2)$ in the above example, 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

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

is a program which realizes generous coordination of P_1 and P_2.

Two programs $P_1 \oplus P_2$ and Q have the same meaning but have different syntax. Then, a question is: which one is more preferable as a coordinated program? An agent may concede in the process of coordination, but the outcome of coordination would be more acceptable if the original information of the agent is reflected to the maximal extent. A desirable property for coordinated programs is that *it should inherit as much information as possible from the original programs*. Comparing $P_1 \oplus P_2$ and Q, sweetness/redness of strawberry, which is included in P_1 or P_2, is kept in $P_1 \oplus P_2$, while no connection between sweetness/redness and strawberry is represented in Q. Thus, in this example, $P_1 \oplus P_2$ is considered preferable to Q.

This intuition is rephrased as follows: when 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? Our solution here is comparing dependency relations between literals. We prefer a coordinated program which inherits dependency relations from the original programs as much as possible. In the above example, $P_1 \oplus P_2$ is considered preferable to Q as a coordinated program, since a dependency relation between *sweet*/*red* and *strawberry* is kept in $P_1 \oplus P_2$ but lost in Q. The reason why we consider dependency relations as a measure of "syntactical closeness" between programs is explained as follows. A logic program is a set of inference rules which represent relations between causes and effects, or conditions and conclusions. If an agent has a belief of causal relations between events, the agent would retain the belief after coordination as far as there is no reason to discard it. So we consider that in the process of coordination it is meaningful to inherit dependency relations in the original programs as much as possible. In what follows, we formalize this idea.

A *(literal) dependency graph* of a program P is a directed graph in which each node represents a ground literal and there is a directed positive edge $\stackrel{+}{\longrightarrow}$ (respectively, negative edge $\stackrel{-}{\longrightarrow}$) from L_1 to L_2 iff there is a ground rule in P such that L_1 appears in the head and L_2 (respectively, *not* L_2) appears in the body of the rule.[3] We say that L_1 *positively depends on* L_2 (respectively, L_1 *negatively depends on* L_2) if $L_1 \stackrel{+}{\longrightarrow} L_2$ (respectively, $L_1 \stackrel{-}{\longrightarrow} L_2$). Let (L_1, L_2) (respectively, $(L_1, not\, L_2)$) be a pair of ground (NAF-)literals such that L_1 positively (respectively, negatively) depends on L_2 in the dependency graph of a program. Let $\delta(P)$ be the collection of such pairs in the dependency graph of P. Then, a condition for selecting preferable coordination is provided as follows.

Definition 5.1. Given two programs P_1 and P_2, suppose two different programs P_3 and P_4 for candidates of coordination. Then, 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, namely, $(S \backslash T) \cup (T \backslash S)$.

[3]A literal dependency graph for EDPs is also introduced in Ben-Eliyahu and Dechter [1994], but it does not consider negative edges.

The above relation represents that P_3 retains dependency relations in P_1 and P_2 more than P_4 does.

Example 5.1. Applying to the introductory example of this section, it becomes

$$\delta(P_1) = \{(sweet, strawberry)\},$$
$$\delta(P_2) = \{(red, strawberry)\},$$
$$\delta(Q) = \emptyset,$$
$$\delta(P_1 \oplus P_2) = \{(sweet, strawberry), (red, strawberry)\}.$$

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 $P_1 \oplus P_2$ is preferable to Q.

Generally, given two sets $\mathcal{AS}(P_1)$ and $\mathcal{AS}(P_2)$, their generous/rigorous coordination can be computed by a series of DNF–CNF conversion. The resulting program is, however, always a set of facts, so that dependency relations in the original programs are completely lost. Dependency relations in $P_1 \oplus P_2$ or $P_1 \otimes P_2$ have the following properties.

PROPOSITION 5.2. *Let P_1 and P_2 be two programs. For literals $L_1, L_2 \in Lit$,*

(1) $(L_1, L_2) \in \delta(P_1) \cup \delta(P_2)$ *implies* $(L_1, L_2) \in \delta(P_1 \oplus P_2)$;
(2) $(L_1, L_2) \in \delta(P_1 \otimes P_2)$ *implies* $(L_1, L_2) \in \delta(P_1) \cup \delta(P_2)$.

PROOF

(1) For every rule $head(r_i) \leftarrow body(r_i)$ in P_i $(i = 1, 2)$, there is a rule $head(r_1)$; $head(r_2) \leftarrow body_*(r_1), body_*(r_2)$ in $P_1 \oplus P_2$. Then, the result holds by $body_*^+(r_i) = body^+(r_i)$.

(2) For every rule $head(r) \cap S \leftarrow body(r)$, *not* $(head(r) \backslash S)$ in $R(P_i, S)$ $(i = 1, 2)$, there is a rule $head(r) \leftarrow body(r)$ in P_i. Hence, the result holds. □

By Proposition 5.2, generous coordination keeps positive dependency relations in the original programs, but negative dependency relations in the original programs may be lost in $P_1 \oplus P_2$ due to the reduction of NAF-literals in $body_*(r_i)$. By contrast, rigorous coordination may eliminate positive/negative dependency relations in the original programs, but it does not introduce new positive dependency relations to $P_1 \otimes P_2$.

Example 5.2. Consider the programs P_1 and P_2 in Example 4.4. The relations $(r, not\ p)$ and $(r, not\ q)$ in $\delta(P_1)$ and $(\neg r, not\ r)$ in $\delta(P_2)$ are eliminated in $\delta(P_1 \otimes P_2)$, while new relations $(p,\ not\ \neg r)$ and $(q,\ not\ \neg r)$ are produced.

In the above example, the eliminated relations do not contribute to the construction of answer sets in $P_1 \otimes P_2$. The newly introduced relations can be eliminated after applying NONMIN to the program $P_1 \otimes P_2$ in this example. Computationally, the DNF–CNF conversion requires exponential computation in general, while $P_1 \oplus P_2$ and $P_1 \otimes P_2$ are computed in polynomial time, given the collections of answer sets of two programs. Thus, the proposed coordinated programs have both representational and computational advantages over the DNF–CNF conversion.

Note that we do not claim here that the proposed coordinated programs provide the *best* solution. We compared our program construction with the DNF–CNF conversion, since it is the simplest way to realize coordination. We used dependency relations for measuring syntactical closeness between programs. However, dependency relations in the original programs are *artificially* kept by introducing extra rules. In the "strawberry" example, adding the rules

$$sweet \leftarrow sweet,\ strawberry,$$
$$red \leftarrow red,\ strawberry,$$

to the DNF–CNF conversion program Q keeps the dependencies (*sweet*, *strawberry*) and (*red*, *strawberry*). In the end, the symmetric difference between this program and the original programs will be the same as the symmetric difference between $P_1 \oplus P_2$ and the original programs.[4] Introducing those tautologies keeps dependencies in the original programs, however, the transformation introduces additional new dependencies: (*sweet*, *sweet*) and (*red*, *red*), which are not in the original program. Moreover, those introduced rules are semantically meaningless and redundant, which are against the consideration of program simplification.

It is worth noting that equivalence-preserving program transformations are often useful to make coordinated programs close to the original programs.

Example 5.3. Consider two programs:

$$P_1: \quad p \leftarrow not\, q,$$
$$P_2: \quad q \leftarrow not\, p.$$

Then, $P_1 \oplus P_2$ becomes

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

and $\Delta(\delta(P_1 \oplus P_2), \delta(P_1) \cup \delta(P_2)) = \{(p,\ not\, q),\ (q,\ not\, p)\}$. Thus, negative dependency relations in the original programs are lost in the generous coordinated program. However, $P_1 \oplus P_2$ is transformed to the semantically equivalent program $tr_1(P_1 \oplus P_2) = P_1 \cup P_2$ 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. More precisely, shifting transforms a disjunctive rule $L_1\,;\cdots;\,L_k \leftarrow Body$ into k nondisjunctive rules: $L_i \leftarrow Body,\ not\, L_1, \ldots, not\, L_{i-1},\ not\, L_{i+1}, \ldots,\ not\, L_k \quad (i = 1, \ldots, k)$. Negative dependency relations lost in generous coordination are possibly revived by applying the shifting transformation tr_1. In this example, it becomes $\Delta(\delta(tr_1(P_1 \oplus P_2)), \delta(P_1) \cup \delta(P_2)) = \emptyset$. Thus, $tr_1(P_1 \oplus P_2)$ is preferable to $P_1 \oplus P_2$. The shifting transformation is applicable to any *head-cycle-free* disjunctive program containing no positive cycle through disjuncts appearing in the head of a disjunctive rule, and preserves the answer set semantics [Ben-Eliyahu and Dechter 1994].

Next, consider two programs:

$$P_3: \quad p\,;\, q \leftarrow,$$
$$P_4: \quad p\,;\, r \leftarrow .$$

[4]This transformation is suggested by a reviewer.

Then, $P_3 \otimes P_4$ becomes

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

and $\Delta(\delta(P_3 \otimes P_4), \delta(P_3) \cup \delta(P_4)) = \{(p, not\, q), (p, not\, r)\}$. Thus, new negative dependency relations are introduced in the rigorous coordinated program. However, $P_3 \otimes P_4$ is transformed to the semantically equivalent program $tr_2(P_3 \otimes P_4) = \{p \leftarrow\}$ by reducing every NAF-literal $not\, L$ such that L appears in the head of no rule. After the transformation, it becomes $\Delta(\delta(tr_2(P_3 \otimes P_4)), \delta(P_3) \cup \delta(P_4)) = \emptyset$. Thus, $tr_2(P_3 \otimes P_4)$ is preferable to $P_3 \otimes P_4$. The reduction tr_2 of NAF-literals, called *positive reduction*, preserves the answer sets of any EDP [Brass and Dix 1997].

6. APPLICATION

Generous or rigorous coordination collects answer sets of multiple programs by taking union/intersection in a simple manner. In multi-agent environments, however, some strategies or preferences are taken into account to select appropriate outcomes. In this section, we consider variants of coordination methods in multi-agent environments.

6.1 Coordination by Majority

When there are more than two agents, it is often the case that the *majority principle* is taken into account.

Example 5.1. John, Lucy and Mary are planning to go to a restaurant together. John wants to have an Asian food—Chinese or Indian or Japanese. Lucy likes Indian or French, and Mary likes Japanese or Italian. Three persons' preferences are encoded as P_1, P_2, and P_3, respectively:

$$P_1: \quad ch\,;\, in\,;\, ja \leftarrow,$$
$$P_2: \quad in\,;\, fr \leftarrow,$$
$$P_3: \quad ja\,;\, it \leftarrow,$$

where $\mathcal{AS}(P_1) = \{\{ch\}, \{in\}, \{ja\}\}$, $\mathcal{AS}(P_2) = \{\{in\}, \{fr\}\}$, and $\mathcal{AS}(P_3) = \{\{ja\}, \{it\}\}$. Rigorous coordination $P_1 \otimes P_2 \otimes P_3$ produces no solution, while generous coordination $P_1 \oplus P_2 \oplus P_3$ have five candidates. Then, they decided to reduce candidates based on the majority principle. As there is no restaurant that is supported by all three persons, they agreed to select restaurants supported by two persons. Consequently, $\{in\}$ and $\{ja\}$ are selected as candidates.

The majority principle is used in our daily life when there is no solution supported by every agent, but there is one supported by a good number of agents. Thus, majority-based coordination is considered useful in practice and it is computed by combining generous/rigorous coordination. When there are n agents, we introduce k-supported coordination which selects answer sets that are common among $k(\leq n)$ or more agents.

Definition 6.1. Let $P_1, \ldots, P_n$ be n programs. Then, *k-supported coordination* ($0 < k \leq n$) (among $P_1, \ldots, P_n$) is defined as a program Π_k such that

$$\mathcal{AS}(\Pi_k) = \{\, S \mid S \in \mathcal{AS}(P'_1) \cap \cdots \cap \mathcal{AS}(P'_k) \,\}$$

where $\{ P'_1, \ldots, P'_k \} \subseteq \{ P_1, \ldots, P_n\}$ and $P'_i \neq P'_j$ for $i \neq j$.

The program Π_k has answer sets that are supported by at least k different agents. In particular, Π_k coincides with rigorous coordination when $k = n$. The majority-based coordination is considered as a special case of k-supported coordination where k is a maximal number such that $\mathcal{AS}(\Pi_k) \neq \emptyset$.

THEOREM 6.1. *Let $P_1, \ldots, P_n$ be n consistent AS-combinable programs. Then, k-supported coordination ($0 < k \leq n$) is computed as*

$$\Pi_k = R_1 \oplus \cdots \oplus R_{\binom{n}{k}},$$

where R_i ($1 \leq i \leq \binom{n}{k}$) is any rigorous coordination among k-combinations of programs from $P_1, \ldots, P_n$ and $R_i \neq R_j$ for $i \neq j$.

PROOF. As $P_1, \ldots, P_n$ are consistent and AS-combinable, $R_1, \ldots, R_{\binom{n}{k}}$ are also consistent and AS-combinable. Then, $S \in \mathcal{AS}(\Pi_k)$
iff $S \in \mathcal{AS}(P'_1) \cap \cdots \cap \mathcal{AS}(P'_k)$ for $\{ P'_1, \ldots, P'_k \} \subseteq \{P_1, \ldots, P_n\}$ and $P'_i \neq P'_j$ for $i \neq j$
iff $S \in \mathcal{AS}(P'_1 \otimes \cdots \otimes P'_k)$ for $\{ P'_1, \ldots, P'_k \} \subseteq \{P_1, \ldots, P_n\}$ and $P'_i \neq P'_j$ for $i \neq j$ (Theorem 4.4)
iff $S \in \mathcal{AS}(R_j)$ for some $1 \leq j \leq \binom{n}{k}$
iff $S \in \mathcal{AS}(R_1 \oplus \cdots \oplus R_{\binom{n}{k}})$ (Theorem 4.2). □

Example 6.2. In Example 5.1, 2-supported coordination among P_1, P_2, and P_3 is computed as follows. First, rigorous coordinations between 2-combinations of programs become:[5]

$$\begin{aligned}
P_1 \otimes P_2 :&\quad in \leftarrow not\ ch,\ not\ ja,\\
&\quad in \leftarrow not\ fr,\\
P_2 \otimes P_3 :&\quad in \leftarrow not\ in,\\
P_3 \otimes P_1 :&\quad ja \leftarrow not\ ch,\ not\ in,\\
&\quad ja \leftarrow not\ it.
\end{aligned}$$

Next, generous coordination among these three programs becomes:

$$\begin{aligned}
(P_1 \otimes P_2) \oplus (P_2 \otimes P_3) \oplus (P_3 \otimes P_1) :&\\
in\,;\, ja &\leftarrow not\ ch,\\
in\,;\, ja &\leftarrow not\ fr,\ not\ it,
\end{aligned}$$

after simplification. The resulting program has two answer sets $\{in\}$ and $\{ja\}$.

[5]Here, "*in*" in $P_2 \otimes P_3$ can be any atom.

6.2 Permissible Coordination

In multi-agent coordination, individual agents often concede one another to reach an agreement. However, an agent may have strong belief that cannot be abandoned and weak belief that can be given up. In another situation, it may happen that one agent is stronger in decision making or more reliable than other agents. In this case, information from particular agents is preferred to information from other ones. To formulate these situations, some preference is taken into account to select information in the process of coordination.

Example 6.3. A company is deciding methods for advertising a new product. A sales person P_1 considers that making CM for TV is most effective. In addition, there is a choice between putting an ad either in newspapers or on the Internet.

$$\begin{aligned} P_1: \quad & np\,;\, net \leftarrow, \\ & tv \leftarrow . \end{aligned}$$

On the other hand, the sales manager P_2 considers an option of putting an ad in either newspapers or magazines. If putting an ad in magazines, another way is making CM for TV. Due to the limited budget, however, putting an ad in newspapers leaves no room for making CM for TV. In this case, putting an ad on the Internet is considered.

$$\begin{aligned} P_2: \quad & tv \leftarrow mag, \\ & net \leftarrow np, \\ & \leftarrow tv,\ np, \\ & np\,;\, mag \leftarrow . \end{aligned}$$

Now two programs have the answer sets: $\mathcal{AS}(P_1) = \{\{tv, np\}, \{tv, net\}\}$ and $\mathcal{AS}(P_2) = \{\{tv, mag\}, \{net, np\}\}$. In this situation, rigorous coordination brings no solution, namely, $P_1 \otimes P_2 = \emptyset$. Then, two persons collect candidates for solutions by generous coordination: $P_1 \oplus P_2 = \{\{tv, np\}, \{tv, net\}, \{tv, mag\}, \{net, np\}\}$. The sales manager first discards $\{tv, np\}$ as it does not satisfy the integrity constraint of P_2. The sales person has a strong belief that making CM for TV is most effective, so $\{net, np\}$ is unsatisfactory for him/her. The manager then takes his/her opinion into account and discards $\{net, np\}$. At this stage, there are two candidates for solutions: $\{tv, net\}$ or $\{tv, mag\}$.

In the above example, answer sets are selected according to several preference criteria. First, the sales manager is in a position higher than the sales person, so that answer sets of P_1 which do not satisfy the integrity constraint of P_2 are discarded. Second, the sales person has a strong belief which cannot be abandoned, so that an answer set of P_2 which does not reflect the belief is eliminated. Such preference conditions are useful to reduce candidates for solutions and to reach final outcome acceptable to each agent.

In rigorous coordination, every answer set included in $P_1 \otimes P_2$ satisfies both P_1 and P_2. In generous coordination, however, an answer set included in $P_1 \oplus P_2$ satisfies either P_1 or P_2, but does not always satisfy both of them. Generous

coordination introduces answer sets of other agents, but an agent would be unwilling to accept some of which do not satisfy its strong beliefs. To retain strong beliefs of each agent, we introduce permissible conditions to generous coordination.

Given a program P, a set PR of *persistent rules* is defined as a subset of P. Intuitively, PR is the set of rules that should be satisfied in the result of generous coordination. In this setting, permissible coordination is defined as follows.

Definition 6.2. Let P_1 and P_2 be two AS-combinable programs, and PR_1 and PR_2 their persistent rules, respectively. A program $P_1 \oplus_P P_2$ is called a *permissible coordination* of P_1 and P_2 if it satisfies the condition

$$\mathcal{AS}(P_1 \oplus_P P_2) = \{ S \mid S \in \mathcal{AS}(P_1) \cup \mathcal{AS}(P_2) \text{ and } S \text{ satisfies } PR_1 \cup PR_2 \}.$$

By the definition, every answer set in the result of permissible coordination satisfies persistent rules of each agent. The permissible coordination reduces to generous coordination when $PR_1 \cup PR_2 = \emptyset$.

Given two AS-combinable programs, permissible coordination is computed by introducing every rule in $PR_1 \cup PR_2$ as an integrity constraint to $P_1 \oplus P_2$. Given a program P, let

$$IC(P) = \{ \leftarrow body(r), \mathit{not\_head}(r) \mid r \in P \}$$

where $\mathit{not\_head}(r)$ is the conjunction of NAF-literals $\{ not\, L_1, \ldots, not\, L_l \}$ for $head(r) = \{ L_1, \ldots, L_l \}$.

LEMMA 6.2 (LIFSCHITZ 1996, PROPOSITION 5.1). *Let P be a program and IC a set of integrity constraints. Then, S is an answer set of P satisfying every constraint in IC iff S is an answer set of $P \cup IC$.*

THEOREM 6.3. *Let P_1 and P_2 be two consistent AS-combinable programs. Then, $\mathcal{AS}(P_1 \oplus_P P_2) = \mathcal{AS}((P_1 \oplus P_2) \cup IC(PR_1) \cup IC(PR_2))$.*

PROOF. By Definition 6.2 and the result of Theorem 4.2, $S \in \mathcal{AS}(P_1 \oplus_P P_2)$ iff S is an answer set of $P_1 \oplus P_2$ and satisfies $PR_1 \cup PR_2$ iff S is an answer set of $P_1 \oplus P_2$ and satisfies $IC(PR_1) \cup IC(PR_2)$ iff $S \in \mathcal{AS}((P_1 \oplus P_2) \cup IC(PR_1) \cup IC(PR_2))$ (Lemma 6.2). □

Example 6.4. Consider two programs P_1 and P_2 in Example 6.3 where $PR_1 = \{ tv \leftarrow \}$ and $PB_2 = \{ \leftarrow tv, np \}$. Then, $(P_1 \oplus P_2) \cup IC(PR_1) \cup IC(PR_2)$ becomes

$$\begin{aligned}
&tv \leftarrow mag,\\
&np\,;\, net\,;\, mag \leftarrow,\\
&tv\,;\, net \leftarrow np,\\
&tv\,;\, np\,;\, mag \leftarrow,\\
&\leftarrow not\, tv,\\
&\leftarrow tv,\ np,
\end{aligned}$$

after simplification. The program has two answer sets $\{tv, net\}$ and $\{tv, mag\}$.

7. RELATED WORK

7.1 Combining Logic Programs

The problem of combining logic programs has been studied by several researchers in different contexts. Baral et al. [1991] introduce algorithms for combining multiple 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:

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

By contrast, $(P_1 \cup IC) \oplus P_2$ in our framework becomes[6]

$$\begin{aligned} & p(x)\,;\, r(a) \leftarrow not\, q(x), \\ & q(b)\,;\, r(a) \leftarrow r(b), \\ & q(a)\,;\, r(a) \leftarrow, \end{aligned}$$

after eliminating tautologies. Comparing two results, the program P_3 has two answer sets $\{p(b), q(a)\}$ and $\{p(b), r(a)\}$, while $(P_1 \cup IC) \oplus P_2$ 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 the algorithms in Baral et al. [1991] 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 (see Bugliesi et al. [1994] and Brogi [2004] for excellent surveys). It combines different programs into one. The problem is then how to provide the meaning of a program in terms of those components. In the presence of negation as failure, Brogi et al. [1999] introduce three meta-level operations for composing normal logic programs: union, intersection, and restriction. The union simply puts two

[6]Here IC is included in P_1 as we handle integrity constraints as a part of a program. The first rule in P_1 is identified with its ground instantiation.

programs together, and the intersection combines two programs by merging a pair of rules with unifiable heads. For instance, given two programs:

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

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. [1997] introduce a variant of the well-founded semantics, and identify conditions for two programs P_1 and P_2 to satisfy the equality $\mathit{Mod}(P_1 \cup P_2) = \mathit{Mod}(P_1) \cap \mathit{Mod}(P_2)$ where $\mathit{Mod}(P)$ is the set of models of P. Etalle and Teusink [1996] consider three-valued completion semantics for program composition as the union of normal open programs. Comparing these 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. By contrast, our goal is to construct a program which has a balanced meaning among different programs, and we do not regard the original program as a subprogram of the coordinated program.

7.2 Merging

Combination of propositional theories has been studied under the names of *merging* [Konieczny and Pino-Pérez 1998] or *arbitration* [Liberatore and Schaerf 1998]. The goal of these research is to provide a new theory which is consistent and preserves as much information as possible from their sources. When two programs P_1 and P_2 do not contradict each other, merging results in $P_1 \cup P_2$. In this regard, merging is different from coordination presented in this paper. For instance, two programs $P_1 = \{ p \leftarrow \}$ and $P_2 = \{ q \leftarrow \}$ are merged into $P_3 = \{ p \leftarrow , \quad q \leftarrow \}$, while 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, beliefs of different agents are mixed together as far as they are consistent, which makes it difficult to distinguish the original beliefs of individual agents after merging. For instance, suppose that 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 $P_5 = \{ p \leftarrow \}$ after eliminating the non-minimal rule in P_4. Later, it turns out that the fact p in P_1 does not hold. At this stage, the agent cannot recover the original information included in P_4 from P_5 alone. By contrast, if generous coordination is done, it becomes $P_4 \oplus P_1 = P_4$ and the original information in P_4 is kept. If there is a good reason to rely on P_1, we can use permissible coordination to prefer P_1 over P_4. In this example, by putting persistent rules as $PR_1 = P_1$, it becomes $P_4 \oplus_P P_1 = \{ p ; q \leftarrow , \quad \leftarrow \mathit{not}\ p \}$ which has the single

answer set $\{p\}$. Note here that the original information in P_4 is still kept in $P_4 \oplus_P P_1$.

Program merging is effective when each agent possesses partial information and problem solving is performed cooperatively. For instance, suppose $P_5 = \{\, p \leftarrow q, r, \quad q \leftarrow\}$ and $P_6 = \{\, p \leftarrow q, r, \quad r \leftarrow\}$. Then, P_5 and P_6 cannot infer p individually, but $P_5 \cup P_6$ can infer p. Our coordination framework is intended to coordinate different belief sets of individual agents, and does not produce new facts inferred cooperatively. In this example, generous coordination becomes $P_5 \oplus P_6 = \{\, p \leftarrow q, r, \quad q\,; r \leftarrow\}$, which does not infer p. In this respect, coordination is more cautious than merging in cooperative reasoning. However, it is worth noting that simple merging does not always produce acceptable conclusions for individual agents in nonmonotonic logic programs. Consider the following example from Gelfond and Lifschitz [1991]. A brave driver crosses railway tracks in the absence of information on an approaching train:

$$cross \leftarrow not\ train.$$

On the other hand, a careful driver crosses railway tracks in the presence of information on no approaching train:

$$cross \leftarrow \neg train.$$

In this example, two rules should not be simply merged because they represent incompatible beliefs of different agents. In fact, simply merging these two programs produces the single solution $\{cross\}$, which is a "brave" solution and would be unacceptable for the careful driver. In our framework, rigorous coordination produces no solution, and generous coordination produces two candidates $\{cross\}$ and $\emptyset$.[7] The example shows that merging nonmonotonic theories does not always produce a consensus among agents, even though they do not contradict one another. When information from different sources conflicts one another, merging employs information supported by the majority of the sources [Konieczny and Pino-Pérez 1998]. Coordination based on the majority principle is also realized as presented in Section 6.1, but the result is basically different from merging as presented above.

7.3 Coordination

Some studies propose multi-agent coordination in logic programming. Buccafurri and Gottlob [2002] 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 fixpoint $JFP(T)$ 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 coordination semantics. For instance, when two programs $P_1 = \{\, p \leftarrow\}$ and $P_2 = \emptyset$ are given, their joint fixpoint semantics becomes $\emptyset$. Interestingly, however, if a tautology $p \leftarrow p$ is added to P_2, the joint fixpoint

[7]Precisely, these two programs are not AS-combinable. Using the transformation in Section 2, however, the second program has the answer set containing $\overline{cross}$.

semantics turns to $\{p\}$. 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 $P_3 = \{\, p \leftarrow p,\ q \leftarrow \}$ 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, rigorous coordination results in $P_1 \otimes P_3 = \emptyset$. If each agent can tolerate accepting belief sets of the other agent, generous coordination produces $P_1 \oplus P_3 = \{\{p\}, \{q\}\}$.

Ciampolini et al. [2003] propose a multi-agent system ALIAS in which each agent has two different knowledge bases written in logic programming. An abductive knowledge base expresses an agent's internal knowledge which is used to achieve the agent's goal. A behavior knowledge base describes desired actions and interactions of the agent with its environment. In particular, two different types of coordination—*collaboration* and *competition*, are realized in its behavioral part. A query specifies behavior of agents to achieve goals, and ALIAS solves problems by interacting two knowledge bases. Coordination in ALIAS is thus procedurally given, which is different from our declarative approach in this article.

De Vos and Vermeir [2004] introduce a system of logic programming agents LPAS. In this system, each agent is represented by an *ordered choice logic program* that has a hierarchical structure expressing preferences among programs. Agents communicate through unidirectional channels and update their own answer sets by incoming information. An agent can defeat incoming information that does not fit its own belief by attaching a higher preference to its own program. The semantics of LPAS is given by an evolutionary fixpoint which is the outcome of communication. A principal difference with our approach is that in LPAS an answer set of one agent changes by incoming information from other agents. This is in contrast to the coordination framework in this article that does not change answer sets of each agent. Our framework has no hierarchical structure over programs, but it can handle preference relations over information as in Section 6.2.

Meyer et al. [2004] 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. In their framework each agent is represented by classical propositional theories, so that those postulates are not generally applied to nonmonotonic theories. In this sense, coordination considered in this article is beside the subject of those postulates. Moreover, their negotiation outcome coincides with the result of merging when two propositional theories are consistent with each other. This is different from our coordinated programs as discussed in Section 7.2. Foo et al. [2005] introduce a theory of multi-agent negotiation in answer set programming. Starting from the initial agreement set $S \cap T$ for an answer set S of an agent and an answer set T of another agent, each agent extends this set to reflect its own demand while keeping consistency with demand of the other agent. When two answer sets S and T do not contradict each other, their algorithm just returns the union $S \cup T$ as the trivial deal. In the "cross-train" example, the algorithm returns $\{\, cross \,\}$ as the solution, which would be unacceptable as stated above.

Sakama and Inoue [2005] propose a method of combining answer sets of different programs. Given two programs P_1 and P_2, they build a program Q satisfying $\mathcal{AS}(Q) = min(\mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2))$ where $\mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2) = \{ S \uplus T \mid \text{for } S \in \mathcal{AS}(P_1) \text{ and } T \in \mathcal{AS}(P_2),\ S \uplus T = S \cup T \text{ if } S \cup T \text{ is consistent; otherwise, } S \uplus T = Lit \}$ and $min(X) = \{ Y \in X \mid \neg \exists Z \in X \text{ such that } Z \subset Y \}$. 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, and extends some answer sets of one program with additional information of another program. Compared with the present work, program composition combines answer sets of each program, and it does not preserve answer sets of the original programs. Moreover, Sakama and Inoue [2006] build a program Q such that $\mathcal{AS}(Q) = min(\{S \cap T \mid S \in \mathcal{AS}(P_1) \text{ and } T \in \mathcal{AS}(P_2)\})$, and a program R such that $\mathcal{AS}(R) = max(\{S \cap T \mid S \in \mathcal{AS}(P_1) \text{ and } T \in \mathcal{AS}(P_2)\})$. The program Q is called *minimal consensus*, and the program R is called *maximal consensus*. Minimal/maximal consensus extracts common beliefs that are included in an answer set of every program, and characterizes minimal/maximal agreements among multiple agents. Coordination, composition, and consensus are all intended to formalize different types of social behaviors of multiple agents in logic programming. A recent study [Inoue and Sakama 2006] reveals that those theories have close relations to a theory of *generalization* in answer set programming.

8. CONCLUSION

This article has studied methods of coordinating multiple logic programs. Two different types of coordination have been introduced and their computational methods have been provided. The majority principle and preference mechanisms were also incorporated to select appropriate outcomes in multi-agent coordination. The proposed theory provides a declarative semantics for coordinating logical agents, which is different from program composition or merging.

In multi-agent systems, coordination could be performed in different ways. In one way, agents have their own knowledge bases and exchange their answer sets. In this case, every agent can share coordination outcomes. In another way, there is a master agent who coordinates answer sets of slave agents. In this case, the master agent has coordination outcomes which are not necessarily shared by slave agents. In both cases, a program having a coordinated semantics serves as a social knowledge base consented among agents.

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. Our coordination methods are implemented on top of the existing answer set solvers like DLV [Eiter et al. 2000]. Given two programs as an input, their answer sets are computed by DLV. Once answer sets are computed, a coordinated program is constructed in polynomial time with respect to the number of rules and the number of answer sets of input programs. The computation would be infeasible when a program possesses an exponential number of answer sets. The same problem, however, arises in computing answer sets by answer set solvers. The

present paper handles ground programs only, so that the output coordinated programs are ground, even when the input programs contain variables. In this respect, further extension is necessary to produce coordinated programs with variables.

There is still room for improvement in computing coordinated programs. For instance, it is hard or even impossible to construct coordinated programs *without* computing answer sets of the original programs in general. However, it might be possible to characterize some special (but not trivial) programs that have simple coordinated programs without computing all answer sets of the original programs. Approximation techniques would be useful in practice, especially in computing the coordination by majority (Section 6.1) which requires combination of coordinated programs. The coordination method proposed in this paper is a step on understanding social behaviors of multiple agents by means of computational logic, and much work remains to be done. In future work, we will refine our framework and also investigate other types of coordination and collaboration as well as their characterization in computational logic.

ACKNOWLEDGMENTS

The authors thank the anonymous referees for their valuable comments on the draft of this article.

REFERENCES

BARAL, C., KRAUS, S., AND MINKER, J. 1991. Combining multiple knowledge bases. *IEEE Trans. Knowl. Data Eng. 3*, 2, 208–220.

BARAL, C. AND GELFOND, M. 1994. Logic programming and knowledge representation. *J. Logic Prog. 19/20*, 73–148.

BEN-ELIYAHU, R. AND DECHTER, R. 1994. Propositional semantics for disjunctive logic programs. *Ann. Math. Artif. Intell. 12*, 1, 53–87.

BRASS, S. AND DIX, J. 1997. Characterizations of the disjunctive stable semantics by partial evaluation. *J. Logic Prog. 32*, 3, 207–228.

BROGI, A., CONTIERO, S. AND TURINI, F. 1999. Programming by combining general logic programs. *J. Logic Comput. 9*, 1, 7–24.

BROGI, A. 2004. On the semantics of logic program composition. In *Program Development in Computational Logic*, Lecture Notes in Computer Science, vol. 3049, Springer-Verlag, New York, 115–151.

BUCCAFURRI, F. AND GOTTLOB, G. 2002. Multiagent compromises, joint fixpoints, and stable models. In *Computational Logic: Logic Programming and Beyond*, Lecture Notes in Artificial Intelligence vol. 2407, Springer-Verlag, New York, 561–585.

BUGLIESI, M., LAMMA, E., AND MELLO, P. 1994. Modularity in logic programming. *J. Logic Prog. 19/20*, 443–502.

CIAMPOLINI, A., LAMMA, E., MELLO, P., TONI, F., AND TORRONI, P. 2003. Cooperation and competition in ALIAS: A logic framework for agents that negotiate. *Ann. Math. Artif. Intell. 37*, 1/2, 65–91.

DE VOS, M. AND VERMEIR, D. 2004. Extending answer sets for logic programming agents. *Ann. Math. Artif. Intell. 42*, 1-3, 103–139.

EITER, T., FABER, W., LEONE, N., AND PFEIFER, G. 2000. Declarative problem solving using the DLV system. In *Logic-Based Artificial Intelligence*, Kluwer, 79–103.

ERDOĞAN, S. T. AND LIFSCHITZ, V. 2004. Definitions in answer set programming. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning*, Lecture Notes in Artificial Intelligence vol. 2923, Springer-Verlag, New York, 114–126.

ETALLE, S. AND TEUSINK, F. 1996. A compositional semantics for normal open programs. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*, MIT Press, Cambridge, MA, 468–482.

FOO, N., MEYER, T., ZHANG, Y. AND ZHANG, D. 2005. Negotiating logic programs. In *Proceedings of the 6th Workshop on Nonmonotonic Reasoning, Action and Change*.

GELFOND, M. AND LIFSCHITZ, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9*, 3/4, 365–385.

INOUE, K. AND SAKAMA, C. 2006. Generality relations in answer set programming. In *Proceedings of the 22nd International Conference on Logic Programming*, Lecture Notes in Computer Science vol. 4079, Springer-Verlag, New York, 211–225.

KONIECZNY, S. AND PINO-PÉREZ, R. 1998. On the logic of merging. In *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning*, Morgan-Kaufmann, San Francisco, CA, 488–498.

LIBERATORE, P. AND SCHAERF, M. 1998. Arbitration (or how to merge knowledge bases). *IEEE Trans. Knowl. Data Eng. 10*, 1, 76–90.

LIFSCHITZ, V. 1996. Foundations of Logic Programming. In *Principles of Knowledge Representation*, G. Brewka, Ed. CSLI Publications, 69–127.

LIFSCHITZ, V. 2002. Answer set programming and plan generation. *Artif. Intell. 138*, 1/2, 39–54.

MEYER, T., FOO, N., KWOK, R. AND ZHANG, D. 2004. Logical foundation of negotiation: outcome, concession and adaptation. In *Proceedings of the 19th National Conference on Artificial Intelligence*, MIT Press, Cambridge, MA, 293–298.

SAKAMA, C. AND INOUE, K. 2004. Coordination between logical agents. In *Proceedings of the 5th International Workshop on Computational Logic in Multi-Agent Systems*, Lecture Notes in Artificial Intelligence vol. 3487, Springer-Verlag, New York, 161–177.

SAKAMA, C. AND INOUE, K. 2005. Combining answer sets of nonmonotonic logic programs. In *Proceedings of the 6th International Workshop on Computational Logic in Multi-Agent Systems*, Lecture Notes in Artificial Intelligence, vol. 3900, Springer-Verlag, New York, 320–339.

SAKAMA, C. AND INOUE, K. 2006. Constructing consensus logic programs. In *Proceedings of the 16th International Symposium on Logic-Based Program Synthesis and Transformation*, Lecture Notes in Computer Science, vol. 4407, 26–42, Springer-Verlag, New York.

VERBAETEN, S., DENECKER, M., AND DE. SCHREYE, D. 1997. Compositionality of normal open logic programs. In *Proceedings of the 1997 International Symposium on Logic Programming*, MIT Press, Cambridge, MA, 371–385.

Received November 2005; revised May 2006; accepted July 2006