Constructing Consensus Logic Programs

Chiaki Sakama[1] and Katsumi Inoue[2]

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

Abstract. In this paper, we suppose an agent which has a knowledge base represented by a logic program under the answer set semantics. We then consider the following two problems: given two programs P_1 and P_2, which have the sets of answer sets $\mathcal{AS}(P_1)$ and $\mathcal{AS}(P_2)$, respectively; (i) find a program Q which has the answer sets as the minimal elements of $\{ S \cap T \mid S \in \mathcal{AS}(P_1)$ and $T \in \mathcal{AS}(P_2) \}$; (ii) find a program R which has the answer sets as the maximal elements of the above set. A program Q satisfying (i) is called *minimal consensus* between P_1 and P_2; and R satisfying (ii) is called *maximal consensus* between P_1 and P_2. Minimal/maximal consensus extracts common beliefs that are included in an answer set of every program. Consensus provides a method of program development under a specification of constructing a program that reflects the meaning of two or more programs. In application, it contributes to a theory of building consensus in multi-agent systems.

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 paper is: what are the suitable conclusions drawn from a *collection* of programs, and how to synthesize a program having such a collective semantics. Those problems become especially important when there exist more than one agent in multi-agent environments. In a multi-agent community, multiple agents generally have different beliefs and intentions. To make decision and act as a whole community, they must seek *consensus* which is acceptable to every individual agent. Suppose a multi-agent system in which each agent has a knowledge base represented by a logic program under the answer set semantics [8]. Answer sets represent sets of literals corresponding to beliefs which can be built by a rational reasoner on the basis of a program [2]. An agent may have (conflicting) alternative sets of beliefs, which are represented by multiple answer sets of a program. Different agents have different collections of answer sets in general. We then capture building consensus among multiple agents as the problem of finding a new program which has consequences common to all programs.

Before formally stating the problem, suppose the following scenario: John and Mary are a couple. John wants to buy a new personal computer. To achieve the goal, he considers two options to save money. The first option is to stop bar-hopping. The second

G. Puebla (Ed.): LOPSTR 2006, LNCS 4407, pp. 26–42, 2007.

one is to give up a family trip this year. These two options are indefinite at the moment. John's belief is represented by the program:

$$\begin{aligned} P_1 : \; & \leftarrow \mathit{not}\ pc, \\ & pc \leftarrow money, \\ & money \leftarrow \neg\, bhop, \\ & money \leftarrow \neg\, trip, \\ & bhop\,;\, \neg\, bhop \leftarrow, \\ & trip\,;\, \neg\, trip \leftarrow, \end{aligned}$$

where ";" represents disjunction and not represents negation as failure. On the other hand, Mary plans to save money to buy her new dress. She also has two options: giving up a family trip or going to no restaurant. She usually does not go to a restaurant. If the family gives up a trip, however, she wants to have a special dinner at a restaurant, instead. She also has indefinite belief on giving up a trip. Mary's belief is represented by the program:

$$\begin{aligned} P_2 : \; & \leftarrow \mathit{not}\ dress, \\ & dress \leftarrow money, \\ & money \leftarrow \neg\, trip, \\ & money \leftarrow \neg\, restaurant, \\ & \neg\, restaurant \leftarrow \mathit{not}\ restaurant, \\ & restaurant \leftarrow \neg\, trip, \\ & trip\,;\, \neg\, trip \leftarrow . \end{aligned}$$

In this situation, P_1 has three answer sets: $S_1 = \{\, pc,\, money,\, \neg\, bhop,\, trip \,\}$, $S_2 = \{\, pc,\, money,\, bhop,\, \neg\, trip \,\}$, and $S_3 = \{\, pc,\, money,\, \neg\, bhop,\, \neg\, trip \,\}$. And P_2 has two answer sets: $T_1 = \{\, dress,\, money,\, \neg\, restaurant,\, trip \,\}$ and $T_2 = \{\, dress,\, money, \neg\, trip,\, restaurant \,\}$. Then, which conclusions should be drawn as consensus of the couple? Since $money$ is included in every answer set of two programs, it seems no doubt to have $\{\, money \,\}$ as a result of consensus. In fact, John and Mary agree to save money, although their purposes are different. On the other hand, $\{\, money,\, trip \,\}$ is a subset of both S_1 and T_1, and $\{\, money,\, \neg\, trip \,\}$ is a subset of S_2, S_3 and T_2. So these two sets are also considered as admissible results of consensus. In the set $\{\, money,\, trip \,\}$, the couple agrees with both saving money and having a trip. In this case, each person considers another way to save money. In the set $\{\, money,\, \neg\, trip \,\}$, the couple agrees with giving up a trip to save money.

This example illustrates that there are two different types of consensus. The first one collects minimal sets of beliefs that are included in an answer set of every program. By contrast, the second one collects maximal sets of beliefs that are included in an answer set of every program. These two types of consensus provide different results in general. The purpose of this paper is to develop a theory of such consensus among multiple logic programs.

Formally, the problems considered in this paper are described as follows:

Given : two programs P_1 and P_2;

Find : (1) a program Q satisfying
$\mathcal{AS}(Q) = min(\{\, S \cap T \mid S \in \mathcal{AS}(P_1)$ and $T \in \mathcal{AS}(P_2)\,\})$;
(2) a program R satisfying
$\mathcal{AS}(R) = max(\{\, S \cap T \mid S \in \mathcal{AS}(P_1)$ and $T \in \mathcal{AS}(P_2)\,\})$,

where $min(X) = \{\, Y \in X \mid \neg\exists Z \in X$ s.t. $Z \subset Y\,\}$ and $max(X) = \{\, Y \in X \mid \neg\exists Z \in X$ s.t. $Y \subset Z\,\}$.

The program Q satisfying (1) is called *minimal consensus* between P_1 and P_2; and the program R satisfying (2) is called *maximal consensus* between P_1 and P_2. We investigate the declarative nature of these two types of consensus, and develop methods for constructing consensus programs from multiple programs.

The rest of this paper is organized as follows. Section 2 presents basic notions used in this paper. Section 3 introduces a framework of consensus among logic programs. Section 4 provides a method for constructing consensus programs. Section 5 addresses applications to multi-agent systems. Section 6 discusses related issues, and Section 7 summarizes the paper.

2 Preliminaries

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

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

where each L_i is a positive/negative literal, i.e., A or $\neg A$ for an atom A, and not is *negation as failure* (NAF). $not\ L$ is called an *NAF-literal*. The symbol ";" represents disjunction. The left-hand side of the rule is the *head*, and the right-hand side is the *body*. For each rule r of the above form, $head(r)$, $body^+(r)$ and $body^-(r)$ denote the sets of literals $\{L_1, \ldots, L_l\}$, $\{L_{l+1}, \ldots, L_m\}$, and $\{L_{m+1}, \ldots, L_n\}$, respectively. Also, $not\_body^-(r)$ denotes the set of NAF-literals $\{not\ L_{m+1}, \ldots, not\ L_n\}$. A disjunction of literals and a conjunction of (NAF-)literals in a rule are identified with its corresponding sets of literals. A rule r is often written as $head(r) \leftarrow body(r)$ where $body(r) = body^+(r) \cup not\_body^-(r)$. A rule r is *disjunctive* if $head(r)$ contains more than one literal. A rule r is an *integrity constraint* if $head(r) = \emptyset$; and r is a *fact* if $body(r) = \emptyset$. A rule r is a *conditional fact* if $body^+(r) = \emptyset$. A program is *NAF-free* if no rule contains NAF-literals. A program is a *kernel form* if it consists of conditional facts only. A program with variables is considered a shorthand for its ground instantiation, and this paper handles ground programs unless stated otherwise.

The semantics of EDPs is given by the *answer set semantics* [8]. Let Lit be the set of all ground literals in the language of a program. A set $S(\subseteq Lit)$ *satisfies* a ground rule r if $body^+(r) \subseteq S$ and $body^-(r) \cap S = \emptyset$ imply $head(r) \cap S \neq \emptyset$. In particular, S satisfies a ground integrity constraint r with $head(r) = \emptyset$ if either $body^+(r) \not\subseteq S$ or $body^-(r) \cap S \neq \emptyset$. S satisfies a ground program P if S satisfies every ground rule in P. Let P be a ground NAF-free EDP. Then, a set $S(\subseteq Lit)$ is an *answer set* of P if S is

a minimal set such that (i) S satisfies P; and (ii) if S contains a pair of complementary literals L and $\neg L$, $S = Lit$. Next, let P be any ground EDP and $S \subseteq Lit$. For every ground rule r in P, the rule $head(r) \leftarrow body^+(r)$ is included in the *reduct* P^S if $body^-(r) \cap S = \emptyset$. Then, S is an *answer set* of P if S is an answer set of P^S. A program has none, one, or multiple answer sets in general. The set of all answer sets of P is written as $\mathcal{AS}(P)$. Note that the collection $\mathcal{AS}(P)$ is an *anti-chain* set, i.e., no element $S \in \mathcal{AS}(P)$ is a proper subset of another element $T \in \mathcal{AS}(P)$. A program having a single answer set is called *categorical* [2]. Categorical programs include important classes of programs such as *definite programs*, *stratified programs*, and *call-consistent programs*. An answer set is *consistent* if it is not Lit. A program P is *consistent* if it has a consistent answer set; otherwise, P is *inconsistent*. An inconsistent program has either no answer set (*incoherent*) or the single answer set Lit (*contradictory*). A literal L is a consequence of *skeptical reasoning* (resp. *credulous reasoning*) in P if L is included in every (resp. some) answer set of P. The set of all literal consequences under skeptical (resp. credulous) reasoning in P is written as $skp(P)$ (resp. $crd(P)$). 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.

Example 2.1. Let P be the program:

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

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

3 Consensus Logic Programs

In this section, we introduce a framework of consensus among multiple programs. 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. Then, define

$$cons(P_1, P_2) = \{\, S \cap T \mid S \in \mathcal{AS}(P_1) \text{ and } T \in \mathcal{AS}(P_2) \,\}.$$

In particular, $cons(P_1, P_2) = \emptyset$ if $\mathcal{AS}(P_1) = \emptyset$ or $\mathcal{AS}(P_2) = \emptyset$.

Definition 3.2. Let P_1 and P_2 be two programs. A program Q is called *minimal consensus* (between P_1 and P_2) if it satisfies the condition

$$\mathcal{AS}(Q) = min(cons(P_1, P_2))$$

where $min(X) = \{\, Y \in X \mid \neg\exists Z \in X \text{ s.t. } Z \subset Y \,\}$. On the other hand, a program R is called *maximal consensus* (between P_1 and P_2) if it satisfies the condition

$$\mathcal{AS}(R) = max(cons(P_1, P_2))$$

where $max(X) = \{\, Y \in X \mid \neg\exists Z \in X \text{ s.t. } Y \subset Z \,\}$. Each element in $\mathcal{AS}(Q)$ (resp. $\mathcal{AS}(R)$) is called a *result* of minimal (resp. maximal) consensus (between P_1 and P_2).

We will often omit "between P_1 and P_2" when it is clear from the context. The above program Q or R is also called a (minimal or maximal) *consensus program*.

Intuitively, a result of minimal consensus represents a minimal agreement. That is, an answer set of Q is a minimal set of beliefs which are included in both an answer set of P_1 and an answer set of P_2. By contrast, a result of maximal consensus represents a maximal agreement. A minimal/maximal consensus is a program which has the meaning as a collection of such minimal/maximal agreement.

Example 3.1. For $\mathcal{AS}(P_1) = \{\{p, s\}, \{q\}\}$ and $\mathcal{AS}(P_2) = \{\{p, t\}, \{r\}\}$, $cons(P_1, P_2) = \{\emptyset, \{p\}\}$. Then, the result of minimal consensus is $\emptyset$, while the result of maximal consensus is $\{\, p \,\}$.

The following properties directly hold by Definition 3.2.

Proposition 3.1. *Let P_1 and P_2 be two programs, Q a minimal consensus, and R a maximal consensus. Consensus programs have the following properties.*

1. *Q and R are consistent iff both P_1 and P_2 are consistent, or one is consistent and the other is contradictory. In particular, if P_1 is contradictory, $\mathcal{AS}(Q) = \mathcal{AS}(R) = \mathcal{AS}(P_2)$.*
2. *Q and R are contradictory iff both P_1 and P_2 are contradictory.*
3. *Q and R are incoherent iff either P_1 or P_2 is incoherent.*

Proposition 3.2. *When two programs P_1 and P_2 are both categorical, minimal and maximal consensus coincide.*

By Proposition 3.1, when one of two programs is inconsistent, the results of consensus are rather trivial. We thus consider consensus of consistent programs hereafter.

Proposition 3.3. *Let P_1 and P_2 be two consistent programs, Q a minimal consensus, and R a maximal consensus. Then,*

1. *$\forall U \in \mathcal{AS}(Q)$, $\exists S \in \mathcal{AS}(P_1)$ and $\exists T \in \mathcal{AS}(P_2)$ such that $U \subseteq S$ and $U \subseteq T$.*
2. *$\forall V \in \mathcal{AS}(R)$, $\exists S \in \mathcal{AS}(P_1)$ and $\exists T \in \mathcal{AS}(P_2)$ such that $V \subseteq S$ and $V \subseteq T$.*
3. *$\forall S \in \mathcal{AS}(P_1)$ and $\forall T \in \mathcal{AS}(P_2)$, $\exists\, U \in \mathcal{AS}(Q)$ such that $U \subseteq S$ and $U \subseteq T$.*

Proof. Since $U = S \cap T$ for some $S \in \mathcal{AS}(P_1)$ and $T \in \mathcal{AS}(P_2)$, $U \subseteq S$ and $U \subseteq T$ hold. Thus, 1 and 2 hold. As U is a minimal element of $cons(P_1, P_2)$, the result 3 follows. □

Proposition 3.3 asserts that a result of minimal/maximal consensus reflects a part of beliefs included in an answer set of every program. Conversely, beliefs included in an answer set of every program are partly reflected as a result of minimal consensus. By contrast, beliefs included in an answer set of a program may not be reflected as a result of maximal consensus.

Example 3.2. In Example 3.1, the result of maximal consensus $\{\, p \,\}$ reflects a part of beliefs in the answer set $\{p, s\}$ of P_1 and a part of beliefs in the answer set $\{p, t\}$ of P_2. But beliefs in the answer set $\{q\}$ of P_1 and $\{r\}$ of P_2 is not reflected as a result of maximal consensus.

Comparing results of minimal consensus and maximal consensus, a result of maximal consensus generally contains more information than a result of minimal consensus. A result of minimal consensus easily becomes an empty set as in Example 3.1.

It may happen that the results of consensus coincide with answer sets of one of the original programs.

Definition 3.3. For two programs P_1 and P_2, let Q be a minimal consensus and R a maximal consensus. When $\mathcal{AS}(Q) = \mathcal{AS}(P_1)$ (resp. $\mathcal{AS}(R) = \mathcal{AS}(P_1)$), P_1 *dominates* P_2 under minimal (resp. maximal) consensus.

Every consistent program dominates contradictory programs under minimal/maximal consensus (Proposition 3.1(1)). When P_1 dominates P_2 under minimal/maximal consensus, we can easily have a consensus program as P_1. The next proposition presents a situation in which such domination happens.

Proposition 3.4. *Let P_1 and P_2 be two consistent programs. Then,*

1. *If $S \subseteq T$ for any $S \in \mathcal{AS}(P_1)$ and for any $T \in \mathcal{AS}(P_2)$, P_1 dominates P_2 under minimal/maximal consensus.*
2. *If $S \subseteq T$ for any $S \in \mathcal{AS}(P_1)$ and for some $T \in \mathcal{AS}(P_2)$, P_1 dominates P_2 under maximal consensus.*

Proof. (1) If $S \subseteq T$ for any $S \in \mathcal{AS}(P_1)$ and for any $T \in \mathcal{AS}(P_2)$, it holds that $cons(P_1, P_2) = \mathcal{AS}(P_1)$. Then, $\mathcal{AS}(Q) = \mathcal{AS}(R) = \mathcal{AS}(P_1)$ and P_1 dominates P_2 under minimal/maximal consensus. (2) If $S \subseteq T$ for any $S \in \mathcal{AS}(P_1)$ and for some $T \in \mathcal{AS}(P_2)$, $S \cap T = S$ holds for such T. For any $T' \in \mathcal{AS}(P_2)$ such that $S \not\subseteq T'$, $S \cap T' \subset S$ holds. As $S \cap T' \subset S \cap T$, $\mathcal{AS}(R) = \mathcal{AS}(P_1)$. Hence, P_1 dominates P_2 under maximal consensus. □

Skeptical/credulous inference in consensus programs has the following properties.

Proposition 3.5. *Let P_1 and P_2 be two consistent programs, Q a minimal consensus, and R a maximal consensus. Then,*

1. $skp(Q) = skp(P_1) \cap skp(P_2)$.
2. $skp(Q) \subseteq skp(R)$.
3. $crd(R) = crd(P_1) \cap crd(P_2)$.
4. $crd(Q) \subseteq crd(R)$.

Proof. 1. For any $L \in Lit$, $L \in skp(Q)$
iff $\forall U \in min(cons(P_1, P_2))$, $L \in U$
iff $\forall V \in cons(P_1, P_2)$, $L \in V$
iff ($\forall S \in \mathcal{AS}(P_1)$, $L \in S$) and ($\forall T \in \mathcal{AS}(P_2)$, $L \in T$)
iff $L \in skp(P_1) \cap skp(P_2)$.

2. For any $L \in Lit$, $L \in skp(Q)$
iff $\forall U \in min(cons(P_1, P_2))$, $L \in U$
only if $\forall V \in max(cons(P_1, P_2))$, $L \in V$
iff $L \in skp(R)$.

3. For any $L \in Lit$, $L \in crd(R)$
iff $\exists U \in max(cons(P_1, P_2))$, $L \in U$
iff $\exists V \in cons(P_1, P_2)$, $L \in V$
iff ($\exists S \in \mathcal{AS}(P_1)$, $L \in S$) and ($\exists T \in \mathcal{AS}(P_2)$, $L \in T$)
iff $L \in crd(P_1) \cap crd(P_2)$.
4. For any $L \in Lit$, $L \in crd(Q)$
iff $\exists U \in min(cons(P_1, P_2))$, $L \in U$
only if $\exists V \in max(cons(P_1, P_2))$, $L \in V$
iff $L \in crd(R)$. □

Thus, minimal consensus extracts skeptical consequences that are common between two programs. By contrast, maximal consensus extracts credulous consequences that are common between two programs. The converse inclusion relations in the second and fourth items of Proposition 3.5 do not hold in general.

Example 3.3. Let $\mathcal{AS}(P_1) = \{\{p, q\}, \{q, r\}\}$ and $\mathcal{AS}(P_2) = \{\{p, q\}\}$ where $skp(P_1) = \{q\}$, $crd(P_1) = \{p, q, r\}$, and $skp(P_2) = crd(P_2) = \{p, q\}$. The result of minimal consensus is $\mathcal{AS}(Q) = \{\{q\}\}$ and the result of maximal consensus is $\mathcal{AS}(R) = \{\{p, q\}\}$. Then, $crd(Q) \subseteq crd(R)$ and $skp(Q) \subseteq skp(R)$.

Proposition 3.5 shows that minimal consensus (resp. maximal consensus) is appropriate for making consensus among skeptical (resp. credulous) reasoners. In building consensus, the problem of interest is the case where one program does not dominate the other and the result of consensus is consistent. In the next section, we present methods for computing consensus programs.

4 Computing Consensus Programs

This section assumes function-free logic programs which are instantiated to finite ground programs. The following transformations are applied to a finite ground EDP P.

- (Elimination of tautologies)
  Delete a rule r from P if $head(r) \cap body^+(r) \neq \emptyset$.
- (Elimination of non-minimal rules)
  Delete a rule r from P if there is another rule r' in P such that $head(r') \subseteq head(r)$, $body^+(r') \subseteq body^+(r)$ and $body^-(r') \subseteq body^-(r)$.
- (Partial evaluation) [3,11] If P contains a rule $head(r) \leftarrow body(r)$ such that $L \in body^+(r)$, and contains r_i $(i = 1, \ldots, n)$ with $L \in head(r_i)$, replace r with the following n rules: $head(r) \cup (head(r_i) \setminus \{L\}) \leftarrow (body(r) \setminus \{L\}) \cup body(r_i)$.
- (Elimination of unfired rules)[1]
  Delete a rule r from P if $head(r') \cap body^+(r) = \emptyset$ for any rule r' in P.

Let $\mathcal{T}(P)$ be a program which is obtained from P by applying one of the transformations presented above. Also, let $\mathcal{T}^{k+1}(P) = \mathcal{T}(\mathcal{T}^k(P))$ $(k \geq 0)$ where $\mathcal{T}^0(P) = P$. Iterative application of transformations reaches a fixpoint $\mathcal{T}^n(P) = \mathcal{T}^{n-1}(P)$ $(n \geq 1)$ and satisfies the following property.

[1] This transformation is considered a special case of partial evaluation in clausal logic [7]. We separate this one here as extended disjunctive programs are outside of clausal logic.

Proposition 4.1. *([3]) Let P be a program, and $\mathcal{T}$ transformations presented above. Then, there is a fixpoint $\mathcal{T}^n(P) = \mathcal{T}^{n-1}(P)$. Moreover, for any such n, $\mathcal{T}^n(P)$ is a kernel form and $\mathcal{AS}(P) = \mathcal{AS}(\mathcal{T}^n(P))$.*

In what follows, a fixpoint $\mathcal{T}^n(P)$, which is in a kernel form, is represented by $ker(P)$.

Definition 4.1. Let P_1 and P_2 be two programs, and $\Sigma \subseteq 2^{Lit}$ an anti-chain over Lit.

1. Compute kernel forms $ker(P_1)$ and $ker(P_2)$.
2. Let $S \in \Sigma$. For any rule $r \in ker(P_1) \cup ker(P_2)$ satisfying $head(r) \cap S \neq \emptyset$, construct a rule r^* such that
   - $head(r^*) = head(r) \cap S$,
   - $body(r^*) = body(r) \cup \{\, not\, L \mid L \in head(r) \setminus S \,\}$
     $\cup \{\, not\, M \mid M \in T \setminus S$ for any $T \in \Sigma \,\}$.

   Put $R(S) = \{r^*\}$ as the set of all such rules.
3. For any $S \in \Sigma$, collect $R(S)$ as $\bigcup_{S \in \Sigma}\ R(S)$.

We define $P_1 \diamond_\Sigma P_2 = \bigcup_{S \in \Sigma}\ R(S)$.

The intuitive meaning of the transformation is as follows. First, kernel forms of P_1 and P_2 are computed by the fixpoint construction of $\mathcal{T}$. In the second step, from $ker(P_1)$ and $ker(P_2)$, any rule r which derives each literal in S is first selected. The rule r is transformed to r^* by (1) restricting disjuncts in the $head(r)$ to those literals appearing in S, (2) literals in $head(r) \setminus S$ are shifted to the body as the set of NAF-literals, and (3) any literal appearing in $T \in \Sigma$ but not in S is appended as the set of NAF-literals. The newly appended NAF-literals do not contribute to the derivation of literals in S. Finally, in the third step, those generated rules r^* are collected as a single program. Note that $P_1 \diamond_\Sigma P_2$ contains no integrity constraint. Integrity constraints do not contribute to producing any element included in a set $S \in \Sigma$. The program $P_1 \diamond_\Sigma P_2$ may contain non-minimal rules which are redundant. In this case, those rules are eliminated according to the second transformation in $\mathcal{T}$.

Example 4.1. Consider two programs:

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

and $\Sigma = \{\{p\}, \{q\}, \{r\}\}$. First, $ker(P_1)$ and $ker(P_2)$ become

$$\begin{aligned} ker(P_1) :\ & p \leftarrow not\, r, \\ & q \leftarrow not\, r, \\ & r \leftarrow not\, q, \\ ker(P_2) :\ & p\,;\, q \leftarrow, \\ & q\,;\, r \leftarrow . \end{aligned}$$

For $\{p\} \in \Sigma$, the first rule in $ker(P_1)$ and the first rule in $ker(P_2)$ are transformed to the same rule:

$$R(\{p\}): \ \ p \leftarrow not\, q,\ not\, r\,.$$

Likewise, for $\{q\} \in \Sigma$ and $\{r\} \in \Sigma$, transformed rules become

$$R(\{q\}): \ q \leftarrow not\, p,\ not\, r\,,$$
$$R(\{r\}): \ r \leftarrow not\, p,\ not\, q\,.$$

As a result, $P_1 \diamond_\Sigma P_2$ becomes

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

The operator $\diamond_\Sigma$ has the following properties.

Proposition 4.2. *The operation $\diamond_\Sigma$ is commutative and associative.*

Proof. The commutative law $P_1 \diamond_\Sigma P_2 = P_2 \diamond_\Sigma P_1$ is straightforward. To see the associative law, both $(P_1 \diamond_\Sigma P_2) \diamond_\Sigma P_3$ and $P_1 \diamond_\Sigma (P_2 \diamond_\Sigma P_3)$ consist of rules r^* build from any $r \in ker(P_1) \cup ker(P_2) \cup ker(P_3)$. As Σ is common, two programs contain the same transformed rules. Hence, $(P_1 \diamond_\Sigma P_2) \diamond_\Sigma P_3 = P_1 \diamond_\Sigma (P_2 \diamond_\Sigma P_3)$. □

The proposition implies that the operation $\diamond_\Sigma$ is confluent; given an anti-chain set Σ and a set of programs, the operation produces the same program independent of the order of computation.

Lemma 4.3. *Let P be a program, and S a consistent answer set of P. Then, for any $L \in Lit$, $L \in S$ iff there is a rule $r \in ker(P)$ such that $L \in head(r)$, $S \cap (head(r) \setminus \{L\}) = \emptyset$, and $S \cap body^-(r) = \emptyset$.*

Proof. Suppose that for some literal $L \in S$, there is no rule $r \in ker(P)$ satisfying the condition. Then, for every rule $r \in ker(P)$, $L \notin head(r)$ or $S \cap (head(r) \setminus \{L\}) \neq \emptyset$ or $S \cap body^-(r) \neq \emptyset$. Thus, for every rule $r \in ker(P)$, $S \cap body^-(r) = \emptyset$ implies either $L \notin head(r)$ or $S \cap (head(r) \setminus \{L\}) \neq \emptyset$ (†). Next, consider the reduct $ker(P)^S$. Since S is an answer set of $ker(P)$ (Proposition 4.1), S is a minimal set satisfying $ker(P)^S$. For any rule $r' \in ker(P)^S$, (†) implies either $L \notin head(r')$ or $S \cap (head(r') \setminus \{L\}) \neq \emptyset$. In this case, $S' = S \setminus \{L\}$ satisfies r'. This contradicts the fact that S is a minimal set satisfying $ker(P)^S$.

Conversely, suppose $L \notin S$ in the presence of a rule $r \in ker(P)$ satisfying the condition. Then, $L \in head(r)$, $S \cap (head(r) \setminus \{L\}) = \emptyset$, and $S \cap body^-(r) = \emptyset$ imply that S does not satisfy r. This contradicts the fact that S satisfies $ker(P)$. □

Lemma 4.4. *Let P_1 and P_2 be two consistent programs, and $\Sigma \subseteq cons(P_1, P_2)$ an anti-chain over 2^{Lit}. Then, $\mathcal{AS}(P_1 \diamond_\Sigma P_2) = \Sigma$.*

Proof. Let $U \in \Sigma$. For any $r^* \in P_1 \diamond_\Sigma P_2$, if $r^* \in R(U')$ for $U' \in \Sigma$ with $U' \neq U$, $body(r^*)$ includes $not\, L$ for $L \in U \setminus U'$. Every such rule is eliminated in $(P_1 \diamond_\Sigma P_2)^U$.

Then, $(P_1 \diamond_\Sigma P_2)^U$ consists of facts: $head(r) \cap U \leftarrow$ where $body^-(r) \cap U = \emptyset$. Thus, U satisfies $(P_1 \diamond_\Sigma P_2)^U$. Suppose that there is a set $V \subseteq U$ which satisfies $(P_1 \diamond_\Sigma P_2)^U$. Then, there is a literal $L \in U \backslash V$. For any fact $head(r^*) \leftarrow$ in $(P_1 \diamond_\Sigma P_2)^U$, $V \cap (head(r^*) \backslash \{L\}) \neq \emptyset$. By $V \subseteq U$, this implies $U \cap (head(r^*) \backslash \{L\}) \neq \emptyset$ (†). On the other hand, by $U = S \cap T$ for some $S \in \mathcal{AS}(P_1)$ and $T \in \mathcal{AS}(P_2)$, $L \in S \cap T$ implies that there is a rule $r_1 \in ker(P_1)$ such that $L \in head(r_1)$, $S \cap (head(r_1) \setminus \{L\}) = \emptyset$, and $S \cap body^-(r_1) = \emptyset$; and there is a rule $r_2 \in ker(P_2)$ such that $L \in head(r_2)$, $T \cap (head(r_2) \backslash \{L\}) = \emptyset$, and $T \cap body^-(r_2) = \emptyset$ (Lemma 4.3). As $U \subseteq S$ and $U \subseteq T$, $U \cap (head(r_1) \setminus \{L\}) = \emptyset$ and $U \cap body^-(r_1) = \emptyset$; and $U \cap (head(r_2) \setminus \{L\}) = \emptyset$ and $U \cap body^-(r_2) = \emptyset$. Such r_1 and r_2 are transformed to r_1^* and r_2^* in $R(U) \subseteq P_1 \diamond_\Sigma P_2$, where $head(r_1^*) = head(r_2^*) = L$; and $body^-(r_1^*) = body^-(r_1) \cup (head(r_1) \setminus \{L\}) \cup \{ M \mid M \in U' \backslash U$ for any $U' \in \Sigma \}$ and $body^-(r_2^*) = body^-(r_2) \cup (head(r_2) \backslash \{L\}) \cup \{ M \mid M \in U' \backslash U$ for any $U' \in \Sigma \}$. By $U \cap body^-(r_1^*) = U \cap body^-(r_2^*) = \emptyset$, $(P_1 \diamond_\Sigma P_2)^U$ includes the fact $L \leftarrow$. Thus, $U \cap (head(r_1^*) \setminus \{L\}) = U \cap (head(r_2^*) \setminus \{L\}) = \emptyset$ for the fact $L \leftarrow$ in $(P_1 \diamond_\Sigma P_2)^U$. This contradicts the assertion (†). Hence, there is no such L, thereby $U = V$. Therefore, U is a minimal set satisfying $(P_1 \diamond_\Sigma P_2)^U$, and an answer set of $P_1 \diamond_\Sigma P_2$.

Conversely, let $U \in \mathcal{AS}(P_1 \diamond_\Sigma P_2)$. Then, for any $L \in U$ there is a rule r^* in $P_1 \diamond_\Sigma P_2$ such that $L \in head(r^*)$, $U \cap (head(r^*) \setminus \{L\}) = \emptyset$, and $U \cap body^-(r^*) = \emptyset$ (Lemma 4.3). As $L \in head(r^*)$ implies $L \in V$ for some $V \in \Sigma$, $L \in U$ implies $L \in V$ for some $V \in \Sigma$ (‡). Suppose that there is no $V \in \Sigma$ such that $U \subseteq V$. Then, for any $V \in \Sigma$, there is a literal $L' \in U \setminus V$. Since L' is included in some $V' \in \Sigma$ (by (‡)), every $r^* \in R(V)$ contains *not* L' in $body(r^*)$. Then, $R(V)^U$ becomes $\emptyset$, so $(P_1 \diamond_\Sigma P_2)^U = \bigcup_{V \in \Sigma} R(V)^U = \emptyset$. Thus, $U \notin \mathcal{AS}(P_1 \diamond_\Sigma P_2)$. Contradiction. Hence, there is $V \in \Sigma$ such that $U \subseteq V$. By the above proof, $V \in \Sigma$ is an answer set of $P_1 \diamond_\Sigma P_2$. Hence, $U \subseteq V$ implies $V \subseteq U$. Therefore, $U = V$ and $U \in \Sigma$. □

Given two programs P_1 and P_2, let Q be a minimal consensus and R a maximal consensus. By Definition 3.2, both $\mathcal{AS}(Q)$ and $\mathcal{AS}(R)$ are anti-chains over 2^{Lit}, and $\mathcal{AS}(Q) \subseteq cons(P_1, P_2)$ and $\mathcal{AS}(R) \subseteq cons(P_1, P_2)$ hold. So we can apply the procedure of Definition 4.1 for $\Sigma = \mathcal{AS}(Q)$ and $\Sigma = \mathcal{AS}(R)$. For notational simplicity, we write $\diamond_{\mathcal{AS}(Q)}$ as $\diamond_Q$, and $\diamond_{\mathcal{AS}(R)}$ as $\diamond_R$.

Theorem 4.5. *Let P_1 and P_2 be two consistent programs. Then, a minimal consensus Q is given as the program $P_1 \diamond_Q P_2$, and a maximal consensus R is given as the program $P_1 \diamond_R P_2$.*

Proof. The results follow from Lemma 4.4. □

The result of Theorem 4.5 presents that the program $P_1 \diamond_Q P_2$ realizes minimal consensus, and $P_1 \diamond_R P_2$ realizes maximal consensus.

Example 4.2. In Example 4.1, Σ is the result of maximal consensus $\mathcal{AS}(R)$ between P_1 and P_2. Hence, $\mathcal{AS}(R) = \mathcal{AS}(P_1 \diamond_R P_2)$ holds.

A consensus program $P_1 \diamond_Q P_2$ or $P_1 \diamond_R P_2$ is constructed inductively by the result of consensus $\mathcal{AS}(Q)$ or $\mathcal{AS}(R)$. The need of a consensus program, in addition to the results of consensus, is explained as follows. A result of consensus represents common

beliefs included in an answer set of every program. However, it brings no information on which the consensus is ground. A consensus program includes sufficient conditions in the original programs to derive beliefs included in the results of consensus. In the next section, we illustrate the use of consensus programs using an example.

By Definition 4.1, the procedure for computing $P_1 \diamond_\Sigma P_2$ includes computation of kernel forms of P_1 and P_2 at the first step, which requires exponential computation in the worst case. Once kernel forms are computed, the second and third steps are executed in time polynomial to $|ker(P_1) \cup ker(P_2)| \times |\Sigma|$, where $|ker(P)|$ represents the number of rules in $ker(P)$ and $|\Sigma|$ represents the number of sets in Σ. In practice, the computation of kernel programs could be done as a compilation process. On the other hand, in runtime environments the computation could be done only for the part of rules that are required to build consensus. The latter case is formally stated below.

Definition 4.2. Let P_1 and P_2 be two programs, and C a minimal/maximal consensus program. Given a set D of literals, a program $C' \subseteq C$ is called a *consensus program with respect to* D if $C' = \{\, r \mid r \in C$ and $head(r) \cap D \neq \emptyset \,\}$.

A consensus program with respect to a particular set D is obtained as

$$\sigma_D(P_1 \diamond_\Sigma P_2) = \{\, r \mid r \in P_1 \diamond_\Sigma P_2 \text{ and } head(r) \cap D \neq \emptyset \,\}$$

where Σ is either $\mathcal{AS}(Q)$ or $\mathcal{AS}(R)$, and $D \cap S \neq \emptyset$ for some $S \in \Sigma$. Generally, $\sigma_D(P_1 \diamond_\Sigma P_2)$ has a computational advantage over $P_1 \diamond_\Sigma P_2$. This is because we are interested in a particular set D of literals, there is no need for computing the whole kernel programs $ker(P_1)$ and $ker(P_2)$ in the first step of Definition 4.1. Instead, for every rule $r \in P_1$ satisfying $head(r) \cap D \neq \emptyset$, iteratively applying the program transformations $\mathcal{T}$ produces the set of conditional facts such that

$$\Gamma_1 = \{\, r \mid head(r) \cap D \neq \emptyset \text{ and } body^+(r) = \emptyset \,\}$$

where $\Gamma_1 \subseteq ker(P_1)$ holds. Likewise, a set $\Gamma_2 (\subseteq ker(P_2))$ of conditional facts is produced by P_2. Then, for such Γ_1 and Γ_2, the procedure of Definition 4.1 is modified as follows.

Definition 4.3. Let P_1 and P_2 be two programs, and $\Sigma \subseteq 2^{Lit}$ an anti-chain over Lit. Given a set D of literals such that $D \cap S \neq \emptyset$ for some $S \in \Sigma$;

1. Compute sets of conditional facts Γ_1 and Γ_2 as presented above.
2. Let $S \in \Sigma$. For any rule $r \in \Gamma_1 \cup \Gamma_2$ satisfying $head(r) \cap S \neq \emptyset$, construct a rule r^* as in Definition 4.1, and put $R'(S) = \{r^*\}$ as the set of all such rules.
3. For any $S \in \Sigma$, collect $R'(S)$ as $\bigcup_{S \in \Sigma} R'(S)$.
   Put $\sigma_D(\bigcup_{S \in \Sigma} R'(S)) = \{\, r \mid r \in \bigcup_{S \in \Sigma} R'(S)$ and $head(r) \cap D \neq \emptyset \,\}$.

Proposition 4.6. $\sigma_D(P_1 \diamond_\Sigma P_2) = \sigma_D(\bigcup_{S \in \Sigma} R'(S))$.

Proof. Any $r^* \in \sigma_D(P_1 \diamond_\Sigma P_2)$ satisfies $head(r^*) \cap D = head(r) \cap S \cap D \neq \emptyset$ for $r \in ker(P_1) \cup ker(P_2)$ and $S \in \Sigma$. On the other hand, any $r^* \in \sigma_D(\bigcup_{S \in \Sigma} R'(S))$ satisfies $head(r^*) \cap D = head(r) \cap S \cap D \neq \emptyset$ for $r \in \Gamma_1 \cup \Gamma_2$ and $S \in \Sigma$. Since $\Gamma_i = \{\, r \mid r \in ker(P_i)$ and $head(r) \cap D \neq \emptyset \,\}$ $(i = 1, 2)$, the result holds. □

Example 4.3. In Example 4.1, let $D = \{p\}$. Then,

$$\begin{aligned} \Gamma_1 :&\ p \leftarrow not\, r\,, \\ \Gamma_2 :&\ p\,;\, q \leftarrow \end{aligned}$$

are obtained as conditional facts with respect to D from P_1 and P_2, respectively. Each rule in $\Gamma_1 \cup \Gamma_2$ is transformed to

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

in $P_1 \diamond_\Sigma P_2$. As a result, $\sigma_D(P_1 \diamond_\Sigma P_2)$ contains the single rule

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

5 Application to Multi-agent Consensus

In multi-agent systems, consensus could be achieved in different ways. In one way, agents have their own knowledge bases and build consensus through communication. In this case, every agent can share the result of consensus. In another way, there is a master agent who coordinates slave agents. In this case, the master agent builds consensus but slave agents are not necessarily share the result of consensus. In both cases, a consensus program serves as a social knowledge base which best reflects belief of individual agents. As shown in Proposition 4.2, the operation $\diamond_\Sigma$ is applied to more than two programs. Thus, minimal/maximal consensus are considered in multi-agent systems in which agents have knowledge bases represented by logic programs. In this section, we give an example to illustrate a process of building consensus among such agents.

Example 5.1. There are three agents, John, Mary, and Susie, who cook dinner together. Each agent has different preference as follows:

1. John wants to have either meat or fish. If he eats meat, he wants to have salad. Else if he eats fish, he wants to have soup. Concerning drinks, he prefers red wine in case of meat, and white wine in case of fish.
2. Mary is vegetarian, so she eats neither meat nor fish. Instead, she wants to have both salad and soup. She likes wine, but no preference between red and white.
3. Susie likes meat and wants to take either salad or soup. She usually drinks beer, but she will give up beer if other two agents agree with drinking red wine. She do not want white wine.

The three agents can communicate on-line. They do not share all their knowledge, but they are informed of results of consensus.

Beliefs of those agents are encoded by the following three logic programs.

$$\begin{aligned} P_j :&\ meat \leftarrow not\, fish, \\ &\ fish \leftarrow not\, meat, \end{aligned}$$

$$
\begin{aligned}
& salad \leftarrow meat,\\
& soup \leftarrow fish,\\
& red \leftarrow meat,\\
& white \leftarrow fish\,,\\
P_m:\ & salad \leftarrow,\\
& soup \leftarrow,\\
& red\,;\, white \leftarrow,\\
& \leftarrow meat,\\
& \leftarrow fish\,,\\
P_s:\ & meat \leftarrow,\\
& salad\,;\, soup \leftarrow,\\
& beer \leftarrow not\ \neg beer,\\
& \neg beer \leftarrow red,\\
& \leftarrow white.
\end{aligned}
$$

Here, P_j, P_m, and P_s correspond to John, Mary, and Susie, respectively. Each program has the answer sets:

$$
\begin{aligned}
\mathcal{AS}(P_j):&\ \{\,meat,\ salad,\ red\,\},\ \ \{\,fish,\ soup,\ white\,\};\\
\mathcal{AS}(P_m):&\ \{\,salad,\ soup,\ red\,\},\ \ \{\,salad,\ soup,\ white\,\};\\
\mathcal{AS}(P_s):&\ \{\,meat,\ salad,\ beer\,\},\ \ \{\,meat,\ soup,\ beer\,\}.
\end{aligned}
$$

In this situation, the result of minimal consensus is the empty set, while the result of maximal consensus is $\{\,salad\,\}$ or $\{\,soup\,\}$. If three agents are credulous reasoners, every agent agrees with cooking either salad or soup. The maximal consensus program $P_j \diamond_R P_m \diamond_R P_s$ then becomes

$$
\begin{aligned}
& salad \leftarrow not\ soup,\\
& soup \leftarrow not\ salad,
\end{aligned}
$$

after eliminating non-minimal rules. The program represents common knowledge agreed by the agents.

The story goes on. Three agents notice that there is no consensus about drink. However, Susie can change her preference if the other two agents agree with drinking red wine. Then, she asks John and Mary to let her know their consensus about drinking. In response to this, John and Mary construct a consensus program with respect to drinking. The maximal consensus results in $\{\{\,salad,\ red\,\},\{\,soup,\ white\,\}\}$. Then, for $D = \{\,red,\ white\,\}$, the maximal consensus program with respect to D becomes $\sigma_D(P_j \diamond_R P_m)$:

$$
\begin{aligned}
& red \leftarrow not\ soup,\ not\ white,\\
& white \leftarrow not\ red,\ not\ salad,
\end{aligned}
$$

after eliminating non-minimal rules. As a result, John and Mary inform Susie of their consensus program with respect to D. Susie then updates her program as $P_s^+ = P_s \cup \sigma_D(P_j \diamond_R P_m)$:

$$\begin{aligned} P_s^+ : \; & meat \leftarrow, \\ & salad\,;\, soup \leftarrow, \\ & beer \leftarrow not\ \neg beer, \\ & \neg beer \leftarrow red, \\ & \leftarrow white, \\ & red \leftarrow not\ soup,\ not\ white, \\ & white \leftarrow not\ red,\ not\ salad. \end{aligned}$$

The updated program has the single answer set: $\{\, meat,\ salad,\ \neg beer,\ red \,\}$. The result of maximal consensus among P_j, P_m and P_s^+ then becomes $\{\, salad,\ red \,\}$. That is, three agents now agree with preparing salad and red wine.

Note that in making the final consensus, the consensus program plays an important role. This is because John and Mary agree on drinking red wine on the condition that they do not take both soup and white wine. The consensus program $\sigma_D(P_j \diamond_R P_m)$ contains this information. So, if Susie took soup, the agreement (drinking red wine) could not be reached. On the other hand, a simple result of consensus, red or $white$, does not bring information on which the consensus is ground. This explains the need of constructing a consensus program, even after obtaining the results of consensus.

6 Related Work

Several studies argue the semantic issue of multiple logic programs. Baral et al. [1] introduce algorithms for combining multiple logic programs. Given two stratified logic programs P_1 and P_2, they produce a program which has an answer set as a subset of an answer set of the program union $P_1 \cup P_2$. Program union or *merging* is the simplest operation for combining different theories. When programs are nonmonotonic, however, merging does not always produce consensus among agents, even though they do not contradict one another. Consider the following example from [8]. 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:

$$cross \leftarrow \neg\, train\,.$$

Simply merging these two programs produces the single solution $\{cross\}$, which would be unacceptable for the careful driver. In our framework, both minimal and maximal consensus produce the empty set.

Brogi et al. [4] introduce meta-level operations for composing normal logic programs. Among them, the intersection operation combines two programs by merging

pair of rules with unifiable heads. For instance, given two programs:

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

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

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

The produced rule specifies information which is common to the original two programs. However, the operation is performed on individual rules, so that resulting rules do not always produce common conclusions. For instance, suppose two programs $P_1 = \{\, p \leftarrow q,\ not\, r, \quad q \leftarrow \,\}$ and $P_2 = \{\, p \leftarrow not\, q,\ r, \quad r \leftarrow \,\}$. Applying intersection operation, the result becomes $P_1 \cap P_2 = \{\, p \leftarrow q,\ r,\ not\, q,\ not\, r \,\}$, which never produces the common conclusion p. By contrast, the consensus program becomes $P_1 \diamond_Q P_2 = P_1 \diamond_R P_2 = \{\, p \leftarrow not\, q, \quad p \leftarrow not\, r \,\}$, which has the single answer set $\{p\}$.

Buccafurri and Gottlob [5] introduce a framework of *compromise logic programs* which aims at reaching common conclusions. Given a collection of programs $T = \{P_1, \ldots, P_n\}$, the *joint fixpoint semantics* of T is defined as the set of minimal elements of $JFP(T) = FP(P_1) \cap \cdots \cap FP(P_n)$ where $FP(P_i)$ is the set of all fixpoints of P_i. For instance, when two programs $P_1 = \{\, p \leftarrow \,\}$ and $P_2 = \{\, p \leftarrow p \,\}$ are given, by $FP(P_1) = \{\{p\}\}$ and $FP(P_2) = \{\emptyset, \{p\}\}$ the joint fixpoint semantics becomes $\{p\}$. Thus, in their framework a tautology $p \leftarrow p$ has a special meaning that "if p is required by another agent, let it be". With this reading, however, $P_1 = \{\, p \leftarrow \,\}$ and $P_3 = \{\, p \leftarrow p, \quad q \leftarrow \,\}$ have the joint fixpoint semantics $\emptyset$, that is, P_3 does not tolerate p when another irrelevant fact q exists in the program. By contrast, the result of minimal/maximal consensus is not affected by the existence of tautologies and the result is $P_1 \diamond_Q P_2 = P_1 \diamond_R P_2 = P_1 \diamond_Q P_3 = P_1 \diamond_R P_3 = \emptyset$.

Consensus is a result of agreement among multiple agents, and the process of reaching agreement is called *negotiation*. Meyer et al. [10] introduce a logical framework for negotiating agents. They introduce two different modes of negotiation: *concession* and *adaptation*. They characterize such negotiation by rational postulates and provide methods for constructing outcomes. In their framework each agent is represented by classical propositional theories, so that those postulates are not generally applied to nonmonotonic theories. Moreover, their negotiation outcome coincides with the result of merging when two propositional theories are consistent with each other. This is different from our results of consensus as discussed above. Foo et al. [6] 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 unsatisfactory as stated above. Wooldridge and Parsons [14] provide conditions for multiple agents to reach an agreement. Given formulas ψ_i $(1 \leq i \leq n)$ as *proposals* made by n agents on the final

round of negotiation, an agreement is reached if $\psi_1 \wedge \cdots \wedge \psi_n$ is satisfiable. Another stronger condition requires convergence to equivalent proposals $\psi_1 \Leftrightarrow \cdots \Leftrightarrow \psi_n$. For instance, given two proposals $\psi_1 = p \vee q$ and $\psi_2 = p \vee \neg q$, the first type of agreement succeeds as $\psi_1 \wedge \psi_2 \equiv p$, while the second type of agreement fails as $\psi_1 \not\equiv \psi_2$. In the context of logic programming, two programs $\{\, p\,;\, q \leftarrow \}$ and $\{\, p\,;\, \neg q \leftarrow \}$ has the result of maximal consensus $\{p\}$ and the result of minimal consensus $\emptyset$. Thus, two types of agreement correspond to different results of consensus. Such correspondence does not hold in general, however. When $\psi_1 = p$ and $\psi_2 = q$, the first type of agreement produces $\psi_1 \wedge \psi_2 \equiv p \wedge q$. The result corresponds to merging two theories and is different from the result of minimal/maximal consensus.

Sakama and Inoue [12] introduce a framework of coordination between logic programs. Given two programs P_1 and P_2, they construct (i) a program Q which has the set of answer sets such that $\mathcal{AS}(Q) = \mathcal{AS}(P_1) \cup \mathcal{AS}(P_2)$; and (ii) a program R which has the set of answer sets such that $\mathcal{AS}(R) = \mathcal{AS}(P_1) \cap \mathcal{AS}(P_2)$. The program Q is called *generous coordination* and R is called *rigorous coordination* of two programs. Compared with the framework of consensus in this paper, generous/rigorous coordination does not change answer sets of the original programs. That is, generous one collects every answer set of each program, while rigorous one picks up answer sets that are common to each program. By contrast, the result of consensus in this paper extracts partial information that are commonly included in an answer set of every program. Thus, coordination and consensus result in different effects, and are used for different purposes. Sakama and Inoue [13] propose a method of combining answer sets of different programs. Given two programs P_1 and P_2, they construct a program Q satisfying $\mathcal{AS}(Q) = min(\mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2))$ where $\mathcal{AS}(P_1) \uplus \mathcal{AS}(P_2) = \{\, S \uplus T \mid \text{for } S \in \mathcal{AS}(P_1) \text{ and } T \in \mathcal{AS}(P_2),\ S \uplus T = S \cup T \text{ if } S \cup T \text{ is consistent; otherwise, } S \uplus T = Lit \,\}$. The program Q 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 by additional information of another program. This is in contrast to our present work which takes intersection of answer sets of two programs. Coordination, composition, and consensus are thus all intended to formalize different types of social behaviors of multiple agents in logic programming. A recent study [9] reveals that those theories have close relations to a theory of generalization in answer set programming.

7 Conclusion

This paper introduced the notion of consensus that extracts common beliefs from answer sets of multiple logic programs. Two different types of consensus, minimal consensus and maximal consensus, were considered, and a method of constructing consensus programs was developed. We applied the framework to building consensus in multi-agent systems. From the viewpoint of program synthesis, construction of consensus programs is considered as a program development under a specification that requests a program reflecting the meanings of two or more programs. In application, it serves as a step on understanding social behaviors of multiple agents by means of logic programming.

The procedure for constructing consensus programs requires computation of all answer sets of programs. This may often be infeasible when a program possesses an exponential number of answer sets. The same problem, however, arises in computing answer sets by existing answer set solvers. In future work, we will refine the present framework and investigate formulation of other types of social behaviors among logic programming agents.

References

1. C. Baral, S. Kraus, and J. Minker. Combining multiple knowledge bases. *IEEE Transactions of Knowledge and Data Engineering*, 3(2):208–220, 1991.
2. C. Baral and M. Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19/20:73–148, 1994.
3. S. Brass and J. Dix. Semantics of (disjunctive) logic programs based on partial evaluation. *Journal of Logic Programming*, 40(1):1–46, 1999.
4. A. Brogi, S. Contiero, and F. Turini. Programming by combining general logic programs. *Journal of Logic and Computation*, 9(1):7–24, 1999.
5. F. Buccafurri and G. Gottlob. Multiagent compromises, joint fixpoints, and stable models. *Computational Logic: Logic Programming and Beyond*, Lecture Notes in Artificial Intelligence 2407, pp. 561–585, Springer, 2002.
6. N. Foo, T. Meyer, Y. Zhang, and D. Zhang. Negotiating logic programs. *Proceedings of the 6th Workshop on Nonmonotonic Reasoning, Action and Change*, 2005.
7. P. A. Gardner and J. C. Shepherdson. Unfold/fold transformations of logic programs. in: J-L. Lassez and G. Plotkin (eds.), *Computational Logic, Essays in Honor of Alan Robinson*, pp. 565-583, MIT Press, 1991.
8. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–385, 1991.
9. K. Inoue and C. Sakama. Generality relations in answer set programming. *Proceedings of the 22nd International Conference on Logic Programming*, Lecture Notes in Computer Science 4079, pp. 211–225, Springer, 2006.
10. T. Meyer, N. Foo, R. Kwok, and D. Zhang. Logical foundation of negotiation: outcome, concession and adaptation. *Proc. AAAI-04*, pp. 293–298, MIT Press, 2004.
11. C. Sakama and H. Seki. Partial deduction in disjunctive logic programming. *Journal of Logic Programming*, 32(3):229–245, 1997.
12. C. Sakama and K. Inoue. Coordination between logical agents. *Proceedings of the 5th International Workshop on Computational Logic in Multi-Agent Systems*, Lecture Notes in Artificial Intelligence 3487, pp. 161–177, Springer, 2005.
13. C. Sakama and K. Inoue. Combining answer sets of nonmonotonic logic programs. *Proceedings of the 6th International Workshop on Computational Logic in Multi-Agent Systems*, Lecture Notes in Artificial Intelligence 3900, pp. 320–339, Springer, 2006.
14. M. Wooldridge and S. Parsons. Languages for negotiation. *Proceedings of the 14th European Conference on Artificial Intelligence*, pp. 393–397, IOS Press, 2000.