Interacting Answer Sets

Chiaki Sakama[1] and Tran Cao Son[2]

[1] Department of Computer and Communication Sciences
Wakayama University, Sakaedani, Wakayama 640-8510, Japan
sakama@sys.wakayama-u.ac.jp
[2] Department of Computer Science
New Mexico State University, Las Cruces, NM 88003, USA
tson@cs.nmsu.edu

Abstract. We consider agent societies represented by logic programs. Four different types of social interactions among agents, *cooperation*, *competition*, *norms*, and *subjection*, are formulated as interactions between answer sets of different programs. Answer sets satisfying conditions of interactions represent solutions coordinated in a multiagent society. A unique feature of our framework is that answer set interactions are specified outside of individual programs. This enables us to freely change the social specifications among agents without the need of modifying individual programs and to separate beliefs of agents from social requirements over them. Social interactions among agents are encoded in a single logic program using constraints. Coordinated solutions are then computed using answer set programming.

1 Introduction

In a multiagent society, agents interact with one another to pursue their goals or perform their tasks. The behavior of one agent is often affected by other agents or constrained in a society he/she belongs to. To reach better states of affairs in a society, goals and behaviors of agents are to be coordinated through agent interactions. Agents interact differently depending on situations. For instance, agents work cooperatively to achieve a common goal, while they behave competitively when their goals are conflicting.

The purpose of this paper is to formulate various types of agent interactions using *answer set programming* (ASP) [1]. In answer set programming, the knowledge base of an agent is represented by a logic program and the belief state of an agent is represented by a collection of answer sets. In the presence of multiple agents, individual agents have their own programs and those programs have different collections of answer sets in general. Consider cooperative problem solving by multiple agents. Each agent has a logic program representing his/her local problem, and computes its answer sets as local solutions. Those solutions are finally integrated to a solution of the global problem in a society. To have successful cooperative problem solving, agents are often required to follow some conditions. We illustrate the situation using an example.

There is a graph G and two robots, say P_1 and P_2, try to cooperatively solve the graph-coloring problem on G. They make a plan such that P_1 paints the left-half of the graph $l(G)$ and P_2 paints the right-half $r(G)$. There are some nodes on the border

$b(G)$ and these nodes can be painted by each robot independently. The robots solve the problem using their logic programs and produce candidate solutions, respectively. At this point, some controls over the behaviors of robots are required.

– Every node on the border must have a unique color. That is, if a node n in the area $b(G)$ is painted with a color c by P_1, the node must also be painted with the same color by P_2, and vice versa.
– Every node which is not on the border must be painted by one of the two robots. That is, each node n in the area $l(G)$ (resp. $r(G)$) is painted by P_1 (resp. P_2) but not by P_2 (resp. P_1).
– Every node in the graph G must be painted by either P_1 or P_2.

These requirements can be expressed as conditions over answer sets of the programs of P_1 and P_2 as follows. Let S be an answer set of a program P_1 and T an answer set of a program P_2. S and T represent local coloring solutions devised by individual robots. The three conditions presented above are rephrased as follows: (i) S contains $paint(n, c)$ iff T contains $paint(n, c)$ for any node n in $b(G)$, (ii) S contains $paint(n, c)$ iff T does not contain $paint(n, c)$ for any node n in $l(G)$ or $r(G)$, (iii) for every node n in G, $paint(n, c)$ must be included in either S or T. Condition (i) represents that the two robots have to *cooperate* to paint nodes lying on the border. By contrast, (ii) represents that nodes in each area are *competitive*, that is, each node in the left-half or the right-half of the graph is painted by only one robot. Condition (iii) represents that painting nodes in the entire graph is *norms* of the two robots. Next consider that each node on the border is painted by two robots, but P_1 is prior to P_2 to make a decision on the color. So if P_1 paints a node n on the border with a color c, then P_2 must accept it. The situation is characterized by changing the condition (i) to implication: (iv) if S contains $paint(n, c)$ for any n in $b(G)$, then T contains $paint(n, c)$. Condition (iv) represents a *subjection* relation between local solutions of two robots.

Cooperation, competition, norms, and subjection are different types of interactions among agents and are frequently used in multiagent systems [15]. To develop multiagent systems in logic programming, the above example illustrates the need of formulating interaction among answer sets to coordinate belief states of multiple agents in a society. The goal of this paper is to provide a computational logic for various types of social interactions among agents. We suppose an agent society in which individual agents have knowledge bases represented by logic programs. Social interactions among agents are then captured as interactions among answer sets of programs. Answer sets satisfying conditions of interactions represent solutions coordinated in a multiagent society. Answer set interactions are extended in various ways and social attitudes of agents are formulated within the framework. Next, by combining different programs into a single joint program, social interactions are specified as constraints over the joint program. Solutions satisfying the requirements of those interactions are then computed as the answer sets of the joint program.

The rest of this paper is organized as follows. Section 2 reviews notions used in this paper. Section 3 formulates different types of interactions between answer sets. The framework is extended in various ways in Section 4. Section 5 provides computation of social interaction in answer set programming. Section 6 discusses related issues, and Section 7 concludes the paper.

2 Preliminaries

In this paper, we consider *extended disjunctive programs* as defined in [7]. An extended disjunctive program (EDP) is a set of *rules* of the form:

$$\ell_1\,;\cdots;\,\ell_l \;\leftarrow\; \ell_{l+1}\,,\ldots,\,\ell_m,\;not\,\ell_{m+1}\,,\ldots,\;not\,\ell_n \quad (n \geq m \geq l \geq 0) \qquad (1)$$

where each ℓ_i is a positive/negative literal. not is *negation as failure* (NAF) and $not\,\ell$ is called an *NAF-literal*. 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 $\{\ell_1,\ldots,\ell_l\}$, $\{\ell_{l+1},\ldots,\ell_m\}$, and $\{\ell_{m+1},\ldots,\ell_n\}$, respectively. A rule r is a *constraint* if $head(r) = \emptyset$; and r is a *fact* if $body^+(r) = body^-(r) = \emptyset$. An EDP is simply called a *program* hereafter. A program P is *NAF-free* if $body^-(r) = \emptyset$ for every rule r in P. A program, rule, or literal is *ground* if it contains no variable. A program containing variables is considered as a shorthand for the set of its ground instances, and this paper handles ground (propositional) programs.

The semantics of an EDP is defined by the *answer set semantics* [7]. Let Lit be the set of all ground literals in the language of a program. Suppose a program P and a set $S\,(\subseteq Lit)$ of ground literals. Then, the *reduct* P^S is the program which contains the ground rule $\ell_1\,;\cdots;\,\ell_l \;\leftarrow\; \ell_{l+1}\,,\ldots,\,\ell_m$ iff there is a ground rule r of the form (1) in P such that $body^-(r) \cap S = \emptyset$. Given an NAF-free EDP P, let S be a set of ground literals which is (i) *closed* under P, i.e., for every ground rule r in P, $body^+(r) \subseteq S$ implies $head(r) \cap S \neq \emptyset$; and (ii) *logically closed*, i.e., it is either consistent or equal to Lit. An *answer set* of an NAF-free program P is a minimal set S satisfying both (i) and (ii). Given an EDP P and a set $S\,(\subseteq Lit)$ of ground literals, 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 $AS(P)$. 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*. Throughout the paper, a program is assumed to be consistent unless stated otherwise.

We suppose an *agent* who has a knowledge base represented by a logic program with the answer set semantics. An agent is often identified with its logic program and we use those terms interchangeably throughout the paper. A *society* is a finite set of agents. We assume that individual agents have their respective programs over a common language and a shared ontology in a society. There are several *interactions* among agents in a society. Among them, we consider the following four interactions which are frequently used in multiagent systems.

Cooperation: an interaction among agents to work together to achieve a common goal.
Competition: an interaction such that a satisfactory result for one agent implies unsatisfactory results for others.
Norms: an interaction that directs an agent to meet expectations or obligations in a society.
Subjection: an interaction that restricts behavior of one agent relative to another agent.

In the next section, we formulate these interactions as well as various social attitudes of agents that would happen during interactions.

3 Answer Set Interactions

In this section, we consider a society that consists of two agents. Interactions between agents are then characterized as the problem of interactions between answer sets of two programs.

3.1 Cooperation

Cooperative agents interact with each other to achieve a common goal. We model the situation by considering that certain facts are held by answer sets of two programs.

Definition 3.1. (cooperation) Let P_1 and P_2 be two programs and $\Phi \subseteq Lit$. Two answer sets $S \in AS(P_1)$ and $T \in AS(P_2)$ *cooperate* on Φ if

$$S \cap \Phi = T \cap \Phi\,. \tag{2}$$

In this case, we also say that S and T make a *cooperation* on Φ.

Condition (2) requires that two answer sets $S \in AS(P_1)$ and $T \in AS(P_2)$ must include the same elements from Φ. This type of interaction is useful to specify agreement or a common goal in a society.

Example 3.1. John and Marry are planning to go to a restaurant. John prefers French and Mary prefers Italian, but they behave together anyway. Programs representing beliefs about restaurants and preferences for John (P_1) and Mary (P_2) are:

$$\begin{array}{ll} P_1 : \ preferred \leftarrow french, & \qquad P_2 : \ preferred \leftarrow italian, \\ \quad\ \ french\,;\,italian \leftarrow . & \qquad\quad\ \ french\,;\,italian \leftarrow . \end{array}$$

P_1 has two answer sets $S_1 = \{\,french, preferred\,\}$ and $S_2 = \{\,italian\,\}$, while P_2 has two answer sets $T_1 = \{\,italian, preferred\,\}$ and $T_2 = \{\,french\,\}$. Putting $\Phi = \{\,french,\ italian\,\}$, we have that S_1 and T_2, and S_2 and T_1 cooperate on Φ.

Cooperation between answer sets is monotonic, namely, cooperation on Φ implies cooperation on its subset.

Proposition 3.1. *(monotonicity) If S and T cooperate on Φ, they cooperate on any Φ' such that $\Phi' \subseteq \Phi$.*

Note that any pair of answer sets cooperates on $\Phi = \emptyset$; and $S = T$ if S and T cooperate on $\Phi = Lit$.

When an answer set S of P_1 includes an answer set T of P_2, S can accept T as a part of beliefs of P_1. By contrast, when S is included in T, S can be extended to adapt to the beliefs of P_2.

Definition 3.2. (accept, adapt) $S \in AS(P_1)$ *accepts* $T \in AS(P_2)$ if $S \supseteq T$. If S accepts T, T *adapts* to S.

Acceptance and adaptation can be characterized by cooperation as follows.

Proposition 3.2. *$S \in AS(P_1)$ accepts $T \in AS(P_2)$ iff S and T cooperate on T. S adapts to T iff S and T cooperate on S.*

Proof. $S \supseteq T$ iff $S \cap T = T$ iff S and T cooperate on T. On the other hand, $S \subseteq T$ iff $S \cap T = S$ iff S and T cooperate on S. □

When $S \in AS(P_1)$ cannot accept nor adapt to $T \in AS(P_2)$, two agents might make a concession.

Definition 3.3. (concession) For any pair of answer sets $S \in AS(P_1)$ and $T \in AS(P_2)$, $\Phi = S \cap T$ is called a *concession* between P_1 and P_2. A *maximal concession* is a concession Φ such that there is no concession Φ' satisfying $\Phi \subset \Phi'$.

Example 3.2. Let $AS(P_1) = \{\{p, q\}, \{r\}\}$ and $AS(P_2) = \{\{p, r\}, \{s\}\}$. Then, $\{p\}$, $\{r\}$ and $\emptyset$ are three possible concessions between P_1 and P_2. Of which the first two sets are maximal concessions.

When there are multiple concessions, a maximal one characterizes a maximal agreement between agents. Concession and cooperation have the following relation.

Proposition 3.3. *If a set Φ is a concession between P_1 and P_2, then there are $S \in AS(P_1)$ and $T \in AS(P_2)$ which cooperate on Φ.*

Proof. If Φ is a concession between P_1 and P_2, then $\Phi = S \cap T$ for some $S \in AS(P_1)$ and $T \in AS(P_2)$. In this case, $S \cap \Phi = T \cap \Phi$ and the result holds. □

3.2 Competition

Competition between agents is a natural phenomenon that, in most cases, results in the satisfaction of one agent and the dissatisfaction of another agent on some issues. We model this phenomenon by considering that the presence (resp. absence) of certain facts in an answer set of one agent demonstrates the satisfaction (resp. dissatisfaction) of the agent with respect to the facts. This leads to the following definition.

Definition 3.4. (competition) Let P_1 and P_2 be two programs and $\Psi \subseteq Lit$. Two answer sets $S \in AS(P_1)$ and $T \in AS(P_2)$ are *competitive* for Ψ if

$$S \cap T \cap \Psi = \emptyset\,. \tag{3}$$

In this case, we also say that S and T are in a *competition* for Ψ.

Condition (3) requires that two answer sets $S \in AS(P_1)$ and $T \in AS(P_2)$ do not share any element belonging to Ψ. This type of interaction is useful to specify a limited resource or an exclusive right in a society.

Example 3.3. John and Mary share a car. John plans to go fishing if he can use the car, while Mary wants to go shopping if the car is available. Programs representing plans for John (P_1) and Mary (P_2) are:

$$\begin{array}{llcll} P_1: & go\_fishing \leftarrow use\_car, & \quad & P_2: & go\_shopping \leftarrow use\_car, \\ & use\_car\,;\, \neg use\_car \leftarrow . & & & use\_car\,;\, \neg use\_car \leftarrow . \end{array}$$

P_1 has two answer sets $S_1=\{go\_fishing,\, use\_car\}$ and $S_2=\{\neg use\_car\}$, while P_2 has two answer sets $T_1 = \{go\_shopping,\, use\_car\}$ and $T_2 = \{\neg use\_car\}$. Putting $\Psi=\{use\_car\}$, we have that S_1 and T_2, S_2 and T_1, and S_2 and T_2 are competitive for Ψ.

The results of competition represent that a successful plan for John implies an unsatisfactory result for Mary, and vice versa.

Proposition 3.4. *(monotonicity) If S and T are competitive for Ψ, they are competitive for any Ψ' such that $\Psi' \subseteq \Psi$.*

As trivial cases, any pair of answer sets is competitive for $\Psi = \emptyset$; and S and T are competitive for Lit if $S \cap T = \emptyset$. Thus interesting cases of competition happen when $\Psi \neq \emptyset$ and $S \cap T \neq \emptyset$.

Definition 3.5. (benefit) Suppose that $S \in AS(P_1)$ and $T \in AS(P_2)$ are competitive for Ψ. Then, S has *benefit* over T wrt Ψ if $S \cap \Psi \neq \emptyset$.

Suppose that S and T are competitive for Ψ. When $S \cap \Psi \supseteq T \cap \Psi$, $S \cap T \cap \Psi = \emptyset$ iff $T \cap \Psi = \emptyset$. This means that in this case there is no chance for T to have benefit over S wrt Ψ. Such a precedence relation in competition is defined as follows.

Definition 3.6. (precedence) Suppose that $S \in AS(P_1)$ and $T \in AS(P_2)$ are competitive for Ψ. Then, S has *precedence* over T wrt Ψ if $S \cap \Psi \supseteq T \cap \Psi$.

Proposition 3.5. *If S has precedence over T wrt Ψ, T cannot have benefit over S wrt Ψ.*

Example 3.4. Suppose that there are two companies P_1 and P_2. P_1 has a right to mine both oil and gas, while P_2 has a right to mine either one of them. The situation is represented by answer sets of programs: $AS(P_1) = \{\{oil, gas\}\}$ and $AS(P_2) = \{\{oil\}, \{gas\}\}$. Then, $\{oil, gas\}$ and $\{gas\}$ are competitive for $\Psi = \{oil\}$, while $\{oil, gas\}$ and $\{oil\}$ are not. In this case, $\{oil, gas\}$ has precedence over $\{gas\}$ wrt $\{oil\}$. This means that if two companies coordinate their answer sets to be competitive for Ψ, there is no chance for P_2 to mine oil.

3.3 Norms

Norms represent expectations or obligations for agents to take some actions. We model the situation by considering that normative goals are included in one of the answer sets of two programs.

Definition 3.7. (norms) Let P_1 and P_2 be two programs and $\Theta \subseteq Lit$. Two answer sets $S \in AS(P_1)$ and $T \in AS(P_2)$ achieve *norms* for Θ if

$$(S \cup T) \cap \Theta = \Theta\,. \tag{4}$$

Condition (4) requires that two answer sets $S \in AS(P_1)$ and $T \in AS(P_2)$ should jointly include every element in Θ. This type of interaction is useful to specify duty or task allocation in a society.

Example 3.5. Mary is planning to have a home party. She asks her friends, John and Susie, to buy wine, juice and water. John will visit a liquor shop and can buy wine or water or both. Susie will visit a grocery store and can buy juice or water or both. Programs representing possible items for shopping by John (P_1) and Susie (P_2) are

$$P_1: \; wine\,;\neg wine \leftarrow, \quad water\,;\neg water \leftarrow . \qquad P_2: \; juice\,;\neg juice \leftarrow, \quad water\,;\neg water \leftarrow .$$

Each program has four answer sets representing buying items. Of which, the following three pairs of answer sets achieve norms for $\Theta = \{\, wine, juice, water\,\}$:
$S_1 = \{wine, water\}$ and $T_1 = \{juice, water\}$; $S_2 = \{wine, \neg water\}$ and $T_2 = \{juice, water\}$; and $S_3 = \{wine, water\}$ and $T_3 = \{juice, \neg water\}$.

Proposition 3.6. *(monotonicity) If S and T achieve norms for Θ, they achieve norms for any Θ' such that $\Theta' \subseteq \Theta$.*

As a special case, any pair of answer sets achieves norms for $\Theta = \emptyset$. To achieve norms, individual agents have their own roles. We formulate this below.

Definition 3.8. (responsible) Let $S \in AS(P_1)$, $T \in AS(P_2)$ and $\Theta \subseteq Lit$. We say that

- S is *individually responsible* for $\Theta \setminus T$;
- S has *no responsibility* if S is individually responsible for $\emptyset$; and
- S is *less responsible* than T if $\Theta \setminus T \subseteq \Theta \setminus S$.

The set $\Theta \setminus T$ is called an *individually responsible set* for S.

Proposition 3.7. *Let $S \in AS(P_1)$, $T \in AS(P_2)$ and $\Theta \subseteq Lit$.*

1. *S and T achieve norms for Θ if either S or T contains its individual responsible set.*
2. *If $S \subseteq T$ then S is less responsible than T.*
3. *If $T \supseteq \Theta$ then S has no responsibility.*

Proof. (1) If $S \supseteq \Theta \setminus T$, $S \cup T \supseteq (\Theta \setminus T) \cup T = \Theta$. Then, $(S \cup T) \cap \Theta = \Theta$. (2) $S \subseteq T$ implies $S \cap \Theta \subseteq T \cap \Theta$, which implies $\Theta \setminus T \subseteq \Theta \setminus S$. (3) $T \supseteq \Theta$ implies $\Theta \setminus T = \emptyset$. □

An individual responsible set $\Theta \setminus T$ in Definition 3.8 represents the least task or obligation for S to achieve given norms. Undertaking individual responsibilities does not always achieve norms, however.

Example 3.6. In Example 3.5, $S_1 = \{wine, water\}$ and $T_1 = \{juice, water\}$ achieve norms for $\Theta = \{\, wine, juice, water\,\}$. Thus, S_1 is individually responsible for $\Theta \setminus T_1 = \{wine\}$ and T_1 is individually responsible for $\Theta \setminus S_1 = \{juice\}$, which means that the individual responsibility of John and Susie is to buy wine and juice respectively. It is easy to see that if John only buys wine and Susie only buys juice then they might not achieve norms for Θ. This is because $S_1' = \{wine, \neg water\}$ and $T_1' = \{juice, \neg water\}$ satisfy the individual responsibility for both John and Susie but do not achieve norms for Θ.

In the above example, John or Susie has to *voluntarily* buy water to achieve the norms. On the other hand, responsibility may change by taking a different pair of answer sets. In Example 3.5, S_2 and T_2 also achieve norms for Θ. But S_2 is responsible for $\Theta \setminus T_2 = \{wine\}$, while T_2 is responsible for $\Theta \setminus S_2 = \{\, juice, water \,\}$. So S_2 and T_2 achieve norms without voluntary actions. This leads us to the following definition.

Definition 3.9. (volunteer) Let $S \in AS(P_1)$, $T \in AS(P_2)$ and $\Theta \subseteq Lit$. We say that S and T *volunteer* for $S \cap T \cap \Theta$. For $S' \in AS(P_1)$ and $T' \in AS(P_2)$, we say that the pair (S, T) requires *less voluntary actions* than (S', T') if $(S \cap T \cap \Theta) \subseteq (S' \cap T' \cap \Theta)$.

By the definition, a voluntary action is required only if $S \cap T \neq \emptyset$.

Proposition 3.8. *Let $\Theta \subseteq Lit$, $\{S, S'\} \subseteq AS(P_1)$ and $\{T, T'\} \subseteq AS(P_2)$ such that S and T (resp. S' and T') achieve norms for Θ. Then, (S, T) requires less voluntary actions than (S', T') iff S and T have more individual responsibility than S' and T'.*

Proof. By $(\Theta \setminus S) \cup (\Theta \setminus T) = \Theta \cap \overline{S \cap T}$ and $(\Theta \setminus S') \cup (\Theta \setminus T') = \Theta \cap \overline{S' \cap T'}$, $(S \cap T \cap \Theta) \subseteq (S' \cap T' \cap \Theta)$ iff $\Theta \cap \overline{S' \cap T'} \subseteq \Theta \cap \overline{S \cap T}$. □

An agent is expected to take a voluntary action in addition to his/her individual responsibility. To declare his/her action to another agent, an agent creates (social) commitment [14].

Definition 3.10. (commitment) A *commitment* $C(P_1, P_2, Q)$ represents a pledge of an agent P_1 to another agent P_2 to realize Q.

Commitments could be canceled, so that $C(P_1, P_2, Q)$ represents a promise of P_1 to P_2 for realizing Q, but it does not necessarily guarantee the outcome of Q.

Proposition 3.9. *$S \in AS(P_1)$ and $T \in AS(P_2)$ achieve norms for Θ only if commitments $C(P_1, P_2, U)$ and $C(P_2, P_1, V)$ are made such that $U \subseteq S$, $V \subseteq T$, and $\Theta \subseteq U \cup V$.*

Proof. $(S \cup T) \cap \Theta = \Theta$ implies the existence of U and V satisfying $U \subseteq S$ and $V \subseteq T$, and $\Theta \subseteq U \cup V$. □

Example 3.7. In order for $S_1 = \{wine, water\}$ and $T_1 = \{juice, water\}$ to achieve norms for $\Theta = \{\, wine, juice, water \,\}$, it is requested to make commitments $C(P_1, P_2, \{wine\})$ and $C(P_2, P_1, \{juice,\ water\})$, for instance.

3.4 Subjection

Subjection represents a situation that the behavior of one agent is dominated by that of another agent. We model the situation by considering that certain facts included in an answer set of one program are included in an answer set of another program.

Definition 3.11. (subjection) Let P_1 and P_2 be two programs and $\Lambda \subseteq Lit$. An answer set $S \in AS(P_1)$ is *subject* to an answer set $T \in AS(P_2)$ wrt Λ if

$$T \cap \Lambda \subseteq S \cap \Lambda\,. \tag{5}$$

In this case, we also say that S and T are in a *subjection* relation wrt Λ.

Condition (5) represents that any element from Λ which is included in an answer set $T \in AS(P_2)$ must be included in an answer set $S \in AS(P_1)$. In other words, S is dominated by T for the selection of elements in Λ. This type of interaction is useful to specify priority or power relations in a society.

Example 3.8. Bob and John are two kids in a family, and they have limited access to the Internet. Since Bob is older than John, any site which is limited to access by Bob is also limited to John, but not vice versa. Now two sites $site_1$ and $site_2$ are considered. Programs representing accessibility to each site by John (P_1) and Bob (P_2) are:

$$\begin{array}{llll} P_1: & acc\_site_1\,;\,\neg acc\_site_1 \leftarrow usr\_John, & P_2: & acc\_site_1\,;\,\neg acc\_site_1 \leftarrow usr\_Bob, \\ & acc\_site_2\,;\,\neg acc\_site_2 \leftarrow usr\_John, & & acc\_site_2\,;\,\neg acc\_site_2 \leftarrow usr\_Bob, \\ & usr\_John \leftarrow . & & usr\_Bob \leftarrow . \end{array}$$

Each program has four answer sets representing accessible sites. Suppose first that the $site_1$ is a site for limited access. Putting $\Lambda_1 = \{\neg acc\_site_1\}$, 12 pairs of answer sets, out of 16 combinations of answer sets of P_1 and P_2, are in subjection relation wrt Λ_1. For instance, the following pairs are two solutions: $S_1 = \{\neg acc\_site_1, \neg acc\_site_2, usr\_John\}$ is subject to $T_1 = \{acc\_site_1, acc\_site_2, usr\_Bob\}$ wrt Λ_1; and $S_2 = \{\neg acc\_site_1, acc\_site_2, usr\_John\}$ is subject to $T_2 = \{\neg acc\_site_1, \neg acc\_site_2, usr\_Bob\}$ wrt Λ_1.

Next, suppose that $site_2$ is added as a site for limited access. Then, Λ_1 is changed to $\Lambda_2 = \{\neg acc\_site_1, \neg acc\_site_2\}$. In this case, there are 9 combinations of answer sets which are in subjection relation wrt Λ_2. For instance, S_1 and T_1 are still in a subjection relation wrt Λ_2, but S_2 and T_2 are not anymore.

Proposition 3.10. *(monotonicity) If S is subject to T wrt Λ, the subjection relation holds for any Λ' such that $\Lambda' \subseteq \Lambda$.*

Proposition 3.11. *If $S \supseteq T$, S is subject to T wrt any Λ.*

If any information in $T \in AS(P_2)$ should be included in $S \in AS(P_1)$, it is achieved by putting $\Lambda = T$.

Proposition 3.12. *If S is subject to T wrt T, $S \supseteq T$.*

Proof. $T \subseteq S \cap T$ implies $S \supseteq T$. □

Note that it is always the case that S is subject to T wrt S as $S \cap T \subseteq S$.

By Definitions 3.1 and 3.6, we have the following relations.

Proposition 3.13. *For any Λ,*

1. *S and T cooperate on Λ iff S is subject to T wrt Λ and T is subject to S wrt Λ.*
2. *If S and T are competitive for Λ and S is subject to T wrt Λ, then S has precedence over T wrt Λ.*

Thus, precedence is considered a special case of a subjection relation.

4 Extensions

4.1 Coordination and Priority

In Section 3 four different types of answer set interactions are introduced. These interactions are combined into a single framework in this section.

Definition 4.1. (coordination) For two programs P_1 and P_2, a tuple of sets of literals $\Omega = (\Phi, \Psi, \Theta, \Lambda)$ is called a *coordination* over P_1 and P_2. Each component X of Ω will be denoted by Ω_X hereafter and is called a *coordination condition* in Ω.

Two answer sets $S \in AS(P_1)$ and $T \in AS(T_2)$ are said to *satisfy* Ω_Φ (resp. Ω_Ψ, Ω_Θ, and Ω_Λ) if they satisfy the conditions in Definition 3.1 wrt Φ (resp. Definition 3.4 wrt Ψ, 3.7 wrt Θ and 3.11 wrt Λ). S and T *satisfy* $C \subseteq \{\Omega_\Phi, \Omega_\Psi, \Omega_\Theta, \Omega_\Lambda\}$, if they satisfy each $X \in C$.

Definition 4.2. (compatible) Let P_1 and P_2 be two programs and Ω a coordination over P_1 and P_2. Two answer sets $S \in AS(P_1)$ and $T \in AS(P_2)$ are *compatible* (or a *solution*) wrt Ω if S and T satisfy Ω_Φ, Ω_Ψ, Ω_Θ, and Ω_Λ.

Since answer set interactions are monotonic with respect to coordination conditions, the compatibility of answer sets is also monotonic, i.e., if S and T are compatible wrt Ω, they are also compatible wrt any $\Omega' = (\Phi', \Psi', \Theta', \Lambda')$ such that $X' \subseteq X$ for $X \in \{\Phi, \Psi, \Theta, \Lambda\}$. This coincides with the intuition that fewer requirements would open the possibility of successful coordination. On the other hand, a tuple Ω specifies different types of social interactions and there may exist conflict among their requirements.

Example 4.1. A company opens positions for a system administrator and a programmer. A system administrator can get a salary higher than a programmer. There are two applicants, P_1 and P_2, who have talents as both an administrator and a programmer. Both P_1 and P_2 share the following knowledge:

$$high\_salary \leftarrow admin,$$
$$low\_salary \leftarrow programmer,$$
$$admin\,;\,programmer \leftarrow .$$

Suppose that two applicants have the same desire to get a higher salary as a system administrator. The common goal is specified by a coordination condition $\Phi = \{high\_salary\}$. However, the company has only one position for an administrator, so the situation is specified by a coordination condition $\Psi = \{admin, programmer\}$. Both P_1 and P_2 have two answer sets: $AS(P_1) = \{S_1, S_2\}$ and $AS(P_2) = \{T_1, T_2\}$ such that $S_1 = T_1 = \{admin, high\_salary\}$ and $S_2 = T_2 = \{programmer, low\_salary\}$. S_1 and T_1 cooperate on Φ, but they are not competitive for Ψ. As a result, no two answer sets of $S \in AS(P_1)$ and $T \in AS(P_2)$ are compatible wrt $\Omega = (\Phi, \Psi, \emptyset, \emptyset)$.

Instead of returning no solution in such cases, we introduce a mechanism of *priorities* over interactions as a method of building a compromised solution. To this end, we assume a preorder relation $\succeq$, called a *priority* relation, over $\{\Omega_\Phi, \Omega_\Psi, \Omega_\Theta, \Omega_\Lambda\}$. Intuitively, $\Omega_x \succeq \Omega_y$ states that satisfying x is more important than satisfying y. A set $C \subseteq \{\Omega_\Phi, \Omega_\Psi, \Omega_\Theta, \Omega_\Lambda\}$ is a *maximal element* wrt $\succeq$ if it satisfies the two conditions: (i) if $x \succeq y$ and $y \in C$ then $x \in C$; and (ii) there exists no $C' \subseteq \{\Omega_\Phi, \Omega_\Psi, \Omega_\Theta, \Omega_\Lambda\}$ such that $C \subset C'$ and C' satisfies (i).

Definition 4.3. (compatible under priority) Let $\succeq$ be a preorder relation defined over $\{\Omega_\Phi, \Omega_\Psi, \Omega_\Theta, \Omega_\Lambda\}$. Two answer sets $S \in AS(P_1)$ and $T \in AS(P_2)$ are *compatible under priority* wrt $(\Omega, \succeq)$ if there exists some maximal element $C \subseteq \{\Omega_\Phi, \Omega_\Psi, \Omega_\Theta, \Omega_\Lambda\}$ wrt $\succeq$ and S and T satisfy C.

In Example 4.1, no pair of answer sets $S \in AS(P_1)$ and $T \in AS(P_2)$ is compatible wrt Ω, but S_1 and T_2 (or S_2 and T_1) are compatible under priority wrt $(\Omega, \{\Omega_\Psi \succeq \Omega_\Phi\})$.

4.2 Dynamic Interactions

In Section 3, Φ, Ψ, Θ, and Λ are given as sets of literals. By specifying them as sets of rules, we can specify interactions that may change depending on different contexts.

Definition 4.4. (dynamic cooperation) Let P_1 and P_2 be two programs and Π a set of rules. Two answer sets $S \in AS(P_1)$ and $T \in AS(P_2)$ make a *weak* (resp. *strong*) *dynamic cooperation* on Π if

$$S \cap X = T \cap X \tag{6}$$

for some (resp. any) answer set X of Π.

Example 4.2. (modified from Example 3.1) John and Mary have two options for dinner, while at lunch John takes hamburger and Mary takes sandwich. The situation is encoded as the program for John (P_1) and the program for Mary (P_2) such that

$$\begin{array}{ll} P_1: & preferred \leftarrow french, \\ & french\,;\,italian \leftarrow dinner, \\ & hamburger \leftarrow lunch, \\ & dinner \leftarrow, \quad lunch \leftarrow . \end{array} \qquad \begin{array}{ll} P_2: & preferred \leftarrow italian, \\ & french\,;\,italian \leftarrow dinner, \\ & sandwich \leftarrow lunch, \\ & dinner \leftarrow, \quad lunch \leftarrow . \end{array}$$

Suppose that Π is given as the set of five rules:

$$\begin{array}{l} french \leftarrow dinner, \\ italian \leftarrow dinner, \\ hamburger \leftarrow lunch, \\ sandwich \leftarrow lunch, \\ lunch\,;\,dinner \leftarrow . \end{array}$$

Π specifies that French and Italian are subject to cooperation for dinner, while hamburger and sandwich are for lunch. Now Π has two answer sets: $\{dinner, french, italian\}$ and $\{lunch, hamburger, sandwich\}$. In this situation, the answer set $\{dinner, french, preferred, lunch, hamburger\}$ of P_1 and the answer set $\{dinner, french,$

$lunch, sandwich\}$ of P_2 make a weak dynamic cooperation on Π. There is another combination of answer sets which make a weak dynamic cooperation on Π (having Italian for dinner), but there is no combination which makes a strong dynamic cooperation on Π.

In the above example, Π specifies cooperations for lunch and dinner, while P_1 and P_2 can cooperate only on dinner. The situation is explained by the existence of a weak dynamic cooperation and the lack of a strong one. Note that if two programs do not contain the fact $lunch \leftarrow$, a strong dynamic cooperation is also possible. Thus, Π specifies cooperations that may change depending on the context of P_1 and P_2. Similar extensions are possible for competition, norms and subjection.

4.3 Interactions among n-Agents

Answer set interactions can be generalized to systems with more than two agents.

Definition 4.5. (AS-interactions among n-agents) Let $P_1, \dots, P_n$ be programs and $S_1 \in AS(P_1), \dots, S_n \in AS(P_n)$ their answer sets. For any collection of k answer sets such that $\{S_{i_1}, \dots, S_{i_k}\} \subseteq \{S_1, \dots, S_n\}$ and $2 \le k \le n$,

1. $S_{i_1}, \dots, S_{i_k}$ make a *k-cooperation* on $\Phi \subseteq Lit$ if
$$S_{i_1} \cap \Phi = \cdots = S_{i_k} \cap \Phi . \tag{7}$$
2. $S_{i_1}, \dots, S_{i_k}$ are in a *k-competition* for $\Psi \subseteq Lit$ if
$$S_{i_1} \cap \cdots \cap S_{i_k} \cap \Psi = \emptyset . \tag{8}$$
3. $S_{i_1}, \dots, S_{i_k}$ achieve *k-norms* for $\Theta \subseteq Lit$ if
$$(S_{i_1} \cup \cdots \cup S_{i_k}) \cap \Theta = \Theta . \tag{9}$$
4. $S_{i_1}, \dots, S_{i_k}$ are in *k-subjection* relations wrt $\Lambda \subseteq Lit$ if
$$S_{i_k} \cap \Lambda \subseteq \cdots \subseteq S_{i_1} \cap \Lambda . \tag{10}$$

We say that $S_1, \dots, S_n$ are *n-compatible* wrt a *coordination* $\Omega = (\Phi, \Psi, \Theta, \Lambda)$ if they satisfy the above four conditions.

Observe that Definition 4.5 reduces to the case of two agents when $n = 2$. It is worth noting that Definition 4.5 has several variants. For instance, we can define a 2-competition for Ψ for the collection of k answer sets as: S_i and S_j $(i \neq j)$ are competitive for Ψ for any pair of answer sets from $\{S_{i_1}, \dots, S_{i_k}\}$. The notion of subjection is extended to the combination of answer sets as: $S_i \cup S_j$ is subject to S_k wrt Λ. Such variants would be also useful, but we do not pursue variants of interactions further here.

4.4 Interactions between Programs

A program generally has more than one answer sets. Then, we can apply the notion of interactions between answer sets to interactions between programs.

Definition 4.6. (interactions between programs) Let P_1 and P_2 be two programs.

1. P_1 and P_2 make a *strong cooperation* (resp. *weak cooperation*) on $\Phi \subseteq Lit$ if
$$S \cap \Phi = T \cap \Phi$$
holds for any (resp. some) pair of answer sets $S \in AS(P_1)$ and $T \in AS(P_2)$.
2. P_1 and P_2 are in a *strong competition* (resp. *weak competition*) for $\Psi \subseteq Lit$ if
$$S \cap T \cap \Psi = \emptyset$$
holds for any (resp. some) pair of answer sets $S \in AS(P_1)$ and $T \in AS(P_2)$.
3. P_1 and P_2 achieve *strong norms* (resp. *weak norms*) for $\Theta \subseteq Lit$ if
$$(S \cup T) \cap \Theta = \Theta$$
holds for any (resp. some) pair of answer sets $S \in AS(P_1)$ and $T \in AS(P_2)$.
4. P_1 and P_2 are in a *strong subjection* (resp. *weak subjection*) relation wrt $\Lambda \subseteq Lit$ if
$$T \cap \Lambda \subseteq S \cap \Lambda$$
holds for any (resp. some) pair of answer sets $S \in AS(P_1)$ and $T \in AS(P_2)$.

No interactions are defined for programs having no answer set.

Strong interactions coincide with weak interactions for two programs each of which has exactly one answer set. For two programs having multiple answer sets, however, strong interactions are very strong conditions and hard to satisfy in general. In fact, examples shown in previous sections are mostly weak interactions. We thus consider that weak interactions would be more useful than strong ones in practice. The above interactions are combined into one as Definition 4.2 and are extended to n-programs as Definition 4.5.

5 Computing Answer Set Interactions

In this section, we provide a method of computing answer set interactions between two programs in ASP.

Definition 5.1. (annotated program) Given a program P_i, its *annotated program* P^i is obtained from P_i by replacing every literal ℓ or NAF-literal $not\,\ell$ appearing in P_i with a new literal ℓ^i or $not\,\ell^i$, respectively.

Example 5.1. Given the program P_1:

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

its annotated program P^1 becomes

$$\begin{aligned}&p^1;\neg q^1 \leftarrow \mathit{not}\, r^1,\\ &r^1 \leftarrow .\end{aligned}$$

Annotations are introduced to distinguish beliefs between different agents. Let us define

$$S^i = \{\, \ell^i \mid \ell \in S \text{ and } S \in AS(P_i)\,\}.$$

The following properties hold.

Proposition 5.1. *Let P_i be a program. S is an answer set of P_i iff S^i is an answer set of P^i.*

Proposition 5.2. *Let P_1 and P_2 be two programs. Then, U is an answer set of $P^1 \cup P^2$ iff $U = S^1 \cup T^2$ for some $S^1 \in AS(P^1)$ and some $T^2 \in AS(P^2)$.*

Proof. As P^1 and P^2 have no literal in common, the result holds by the *splitting set theorem* of [9]. □

Next we provide specification of social interactions in a program.

Definition 5.2. (social constraints) Let P_1 and P_2 be two programs and Ω=($\Phi, \Psi, \Theta, \Lambda$) a coordination over P_1 and P_2. Social interactions (2), (3), (4), and (5) between P_1 and P_2 are specified as a set SC of *social constraints* as follows.

1. For each $\ell \in \Phi$, SC contains a pair of constraints:

$$\leftarrow \ell^1,\ \mathit{not}\,\ell^2, \tag{11}$$
$$\leftarrow \ell^2,\ \mathit{not}\,\ell^1. \tag{12}$$

2. For each $\ell \in \Psi$, SC contains a constraint:

$$\leftarrow \ell^1, \ell^2\,. \tag{13}$$

3. For each $\ell \in \Theta$, SC contains a constraint:

$$\leftarrow \mathit{not}\,\ell^1,\ \mathit{not}\,\ell^2\,. \tag{14}$$

4. For each $\ell \in \Lambda$, SC contains a constraint:

$$\leftarrow \ell^2,\ \mathit{not}\,\ell^1\,. \tag{15}$$

The program $P^1 \cup P^2 \cup SC$ is called a *joint program*.

Constraints (11) and (12) represent that the presence of any literal $\ell \in \Phi$ in an answer set of P_1 forces the presence of the same literal in an answer set of P_2, and the other way round. The constraint (13) indicates that any literal $\ell \in \Psi$ cannot belong to an answer set of P_1 and an answer set of P_2 at the same time. By contrast, the constraint (14) expresses that every literal $\ell \in \Theta$ must belong to either an answer set of P_1 or an answer set of P_2. Finally, the constraint (15) says that every literal $\ell \in \Lambda$ in an answer set of P_2 must belong to an answer of P_1.

With this setting, the next theorem holds.

Theorem 5.3. *Let P_1 and P_2 be two programs and $\Omega = (\Phi, \Psi, \Theta, \Lambda)$ a coordination over P_1 and P_2. Two answer sets $S \in AS(P_1)$ and $T \in AS(P_2)$ are compatible wrt Ω iff $S^1 \cup T^2$ is an answer set of the joint program $P^1 \cup P^2 \cup SC$.*

Proof. By Proposition 5.2, U is an answer set of $P^1 \cup P^2$ iff $U = S^1 \cup T^2$ for some $S^1 \in AS(P^1)$ and some $T^2 \in AS(P^2)$. Then, U is an answer set of $P^1 \cup P^2 \cup SC$ iff $\ell^1 \in S^1$ and $\ell^2 \in T^2$ satisfy the constraints SC for any ℓ in each coordination condition in Ω. In this case, S and T are compatible wrt Ω. Hence, the result holds. □

Theorem 5.4. *Let P_1 and P_2 be two programs and $\Omega = (\Phi, \Psi, \Theta, \Lambda)$ a coordination over P_1 and P_2. Deciding whether there are two answer sets $S \in AS(P_1)$ and $T \in AS(P_2)$ that are compatible wrt Ω is Σ_2^P-complete.*

Proof. Deciding the existence of an answer set of an EDP is Σ_2^P-complete [5], hence the result holds by Theorem 5.3. □

The notion of joint programs is extended to n-agents in a straightforward manner, and compatible answer sets among n-agents are computed accordingly.

6 Discussion

6.1 Answer Set Interactions = Answer Sets + Control

In this paper, interactions among answer sets are specified *outside* of individual programs. One may wonder that such an extra mechanism is really needed to encode the specification. In Example 3.1, for instance, the set Φ could be specified inside of each program as

$$\begin{aligned} P_1' : \; & preferred\_by\_john \leftarrow john\_go\_french, \\ & john\_go\_french \leftarrow mary\_go\_french, \\ & john\_go\_italian \leftarrow mary\_go\_italian, \\ & john\_go\_french \,;\, john\_go\_italian \leftarrow, \end{aligned}$$

and

$$\begin{aligned} P_2' : \; & preferred\_by\_mary \leftarrow mary\_go\_italian, \\ & mary\_go\_french \leftarrow john\_go\_french, \\ & mary\_go\_italian \leftarrow john\_go\_italian, \\ & mary\_go\_french \,;\, mary\_go\_italian \leftarrow . \end{aligned}$$

In this case, $P_1' \cup P_2'$ has two answer sets $\{\, john\_go\_french,\ mary\_go\_french,$ $preferred\_by\_john \,\}$ and $\{\, john\_go\_italian,\ mary\_go\_italian,\ preferred\_by\_$ $mary \,\}$. These two answer sets correspond to two possible results of cooperation.

There are mainly two reasons why we do not take this solution in this paper. First, programs P_1 and P_2 represent beliefs of individual agents, while a coordination $\Omega = (\Phi, \Psi, \Theta, \Lambda)$ represents social requirements over them. Such a separation has an advantage of not only reducing codes of individual programs but specifying interactions independent of individual programs. For instance, social requirements may change in time as in Example 3.8. If social interactions are encoded in programs, programs are to be updated whenever situation changes. Thanks to the separation of Ω from individual programs, any change in Ω does not affect the content of programs. The separation of two components also accords to the principle of "*Algorithm = Logic + Control*" by Kowalski [8]. In fact, Ω represents control over answer sets. In this sense, answer set interactions are considered answer sets of different programs plus control over them.

Second, as remarked in [13], simply merging nonmonotonic logic programs does not always produce acceptable conclusions for individual agents. Consider the following situation [7]. A brave driver crosses railway tracks in the absence of information on an approaching train:

$$P_1: \quad cross \leftarrow not\ train.$$

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

$$P_2: \quad cross \leftarrow \neg train.$$

Simply merging these two programs $P_1 \cup P_2$ produces the single answer set $\{cross\}$, which is a "brave" solution and would be unacceptable for the careful driver. The example shows that merging nonmonotonic theories does not always produce an agreement among agents, even though they do not contradict one another. Note that a joint program in Definition 5.2 also merges two programs, but the problem does not happen as P^1 and P^2 contain different annotated literals. Also it should be noted that a joint program is introduced not for merging beliefs of agents but for computing interactions among agents. Given individual programs and a coordination Ω over them, they are compiled into a single joint program to compute solutions of the coordination.

We provide different forms of interactions and various social attitudes of agents, but one may not fully agree with definitions given in this paper. In fact, "there is no universally accepted definition of agency or of intelligence" [15]. Our intention is not to provide universally accepted definitions of agent interactions, but to turn ill-defined agents problems to a well-defined semantic problem in computational logic. Answer set interactions have clear semantics, which are simple yet useful for answer set based agent programming. Moreover, interactions are defined as set theoretic relations, so that similar notions are defined for any model theoretic semantics of computational logics or logic programs.

6.2 Related Work

There are several studies which provide logics for social interactions among agents. Meyer et al. [10] introduce a logical framework for negotiating agents. They introduce two different modes of negotiation: *concession* and *adaptation*. Concession weakens an initial demand of an agent, while adaptation expands an initial demand to accommodate a demand of another agent. In [10], concession and adaptation change original theories of two agents only when they contradict each other. Those definitions are thus different from ours of Definitions 3.2 and 3.3. They provide rational postulates to characterize negotiated outcomes between two agents, and describe 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 the context of logic programming, Buccafurri and Gottlob [2] introduce a framework of *compromise logic programs*. Given a collection of programs $T=\{Q_1,\ldots,Q_n\}$, the *joint fixpoint semantics* of T is defined as the set of minimal elements in $JFP(T) = FP(Q_1)\cap\cdots\cap FP(Q_n)$ where $FP(Q_i)$ is the set of all fixpoints of Q_i. The goal of their study is providing common conclusions among different programs. Ciampolini et al. [4] propose *abductive logic agents* (ALIAS) in which two different types of coordination, *collaboration* and *competition*, are realized. A query specifies behaviors of agents to achieve goals, and ALIAS solve the goal by communicating agents. In ALIAS, coordination is operationally given using inference rules, which is different from our declarative specifications in this paper. Buccafurri and Caminiti [3] introduce a *social logic program* (SOLP) which has rules of the form: $head \leftarrow [selection\_condition]\{body\}$, where $selection\_condition$ specifies social conditions concerning either the cardinality of communities or particular individuals satisfying the body. Agent interactions are thus encoded in individual programs in SOLP, which is in contrast to our approach of separating beliefs of agents and social interactions among them. Foo et al. [6] introduce a theory of multiagent 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. Their algorithm returns new programs having answer sets which are consistent with each other and keep the agreement set. Sakama and Inoue [11] propose a method of combining answer sets of different logic programs. Given two programs P_1 and P_2, they build a program Q satisfying $AS(Q) = min(\{S \cup T \mid S \in AS(P_1) \text{ and } T \in AS(P_2)\})$, which they call a *composition* of P_1 and P_2. Sakama and Inoue [12] also build a *minimal consensus* program Q satisfying $AS(Q) = min(\{S \cap T \mid S \in AS(P_1) \text{ and } T \in AS(P_2)\})$, and a *maximal consensus* program R satisfying $AS(R) = max(\{S \cap T \mid S \in AS(P_1) \text{ and } T \in AS(P_2)\})$. Composition extends one's beliefs by combining answer sets of two programs, while consensus extracts common beliefs from answer sets of two programs. Different from our approach, studies [6,11,12] *change* answer sets of the original programs for coordination results. Sakama and Inoue [13] introduce two notions of coordination between programs. A *generous coordination* constructs a program Q which has the set of answer sets such that $AS(Q) = AS(P_1) \cup AS(P_2)$, while a *rigorous coordination* constructs a program R which has the set of answer sets such that $AS(R) = AS(P_1) \cap AS(P_2)$. These two coordination methods just take the union or intersection of the collections of answer sets of two programs, and do not take extra coordination conditions into account as we do in this paper.

7 Conclusion

In this paper, we introduced the notion of answer set interactions and used it to characterize different types of interactions between agents represented as logic programs. Among other things, we considered cooperation, competition, norms, and subjection between agents. Each of these interactions can be viewed as a constraint on the collection of answer sets of the involving agents. The main advantage of this approach to specifying interactions between agents lies in its flexibility, i.e., interactions between agents are specified outside of individual agents' programs. We also discussed a possible way for computing coordinated solutions using answer set programming.

Several issues remain for further research. One such issue is the extension of answer set interactions to consider other forms of interactions between agents (e.g., resource constraints). We would also like to investigate possible ways to integrate this notion into multiagent planning. Another issue is realizing agent interactions that may evolve, such as negotiation, by incorporating belief update that may arise during interaction. For the implementation of our approach, the method of computing interactions proposed in this paper supposes situations where centralized control over all agents is possible. Typical centralized control is found in the master-slave architecture of MAS. On the other hand, it is also important to develop a framework for specifying and computing answer set interactions in a distributed setting. The issue is left for future work.

Acknowledgment. The second author is partially supported by the NSF grants IIS-0812267 and CREST-0420407.

References

1. Baral, C.: Knowledge Representation, Reasoning, and Declarative Problem Solving. Cambridge University Press, Cambridge (2003)
2. Buccafurri, F., Gottlob, G.: Multiagent compromises, joint fixpoints, and stable models. In: Kakas, A.C., Sadri, F. (eds.) Computational Logic: Logic Programming and Beyond. LNCS (LNAI), vol. 2407, pp. 561–585. Springer, Heidelberg (2002)
3. Buccafurri, F., Caminiti, G.: A social semantics for multi-agent systems. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS (LNAI), vol. 3662, pp. 317–329. Springer, Heidelberg (2005)
4. Ciampolini, A., Lamma, E., Mello, P., Toni, F., Torroni, P.: Cooperation and competition in ALIAS: a logic framework for agents that negotiate. Annals of Mathematics and Artificial Intelligence 37, 65–91 (2003)
5. Eiter, T., Gottlob, G.: Complexity results for disjunctive logic programming and application to nonmonotonic logics. In: Miller, D. (ed.) Proceedings of the 1993 International Symposium on Logic Programming, pp. 266–278. MIT Press, Cambridge (1993)
6. Foo, N., Meyer, T., Zhang, Y., Zhang, D.: Negotiating logic programs. In: Proceedings of the 6th Workshop on Nonmonotonic Reasoning, Action and Change (2005)
7. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing 9, 365–385 (1991).
8. Kowalski, R.A.: Algorithm = Logic + Control. Communications of the ACM 22, 424–436 (1979)

9. Lifschitz, V., Turner, H.: Splitting a logic program. In: Hentenryck, P.V. (ed.) Proceedings of the 11th International Conference on Logic Programming, pp. 23–37. MIT Press, Cambridge (1994)
10. Meyer, T., Foo, N., Kwok, R., Zhang, D.: Logical foundation of negotiation: outcome, concession and adaptation. In: Proceedings of the 19th National Conference on Artificial Intelligence, pp. 293–298. MIT Press, Cambridge (2004)
11. Sakama, C., Inoue, K.: Combining answer sets of nonmonotonic logic programs. In: Toni, F., Torroni, P. (eds.) CLIMA 2005. LNCS (LNAI), vol. 3900, pp. 320–339. Springer, Heidelberg (2006)
12. Sakama, C., Inoue, K.: Constructing consensus logic programs. In: Puebla, G. (ed.) LOPSTR 2006. LNCS, vol. 4407, pp. 26–42. Springer, Heidelberg (2007)
13. Sakama, C., Inoue, K.: Coordination in answer set programming. ACM Transactions on Computational Logic 9, Article No.9 (2008); Shorter version: Coordination between logical agents. In: Leite, J., Torroni, P. (eds.) CLIMA 2004. LNCS (LNAI), vol. 3487, pp. 161–177. Springer, Heidelberg (2005)
14. Singh, M.P.: An ontology for commitments in multiagent systems: toward a unification of normative concepts. Artificial Intelligence and Law 7, 97–113 (1999)
15. Weiss, G. (ed.): Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press, Cambridge (1999)