Multi-Agent Compromises and Joint Fixpoint Semantics

Francesco Buccafurri[1] and Georg Gottlob[2]

[1] DIMET, Universitá di Reggio Calabria, I-89100 Reggio Calabria, Italy,
bucca@ing.unirc.it
[2] Institut für Informationssysteme, Technische Universität Wien, A-1040 Wien, Austria,
gottlob@dbai.tuwien.ac.at

Abstract. We assume the requirements or desires of an agent are modeled by a logic program. In a multi-agent setting, a joint decision of the agents, reflecting a compromise of the various requirements, corresponds to a suitable joint model of the respective logic programs. In this paper, an appropriate semantics for selecting joint models representing compromises is proposed: *the joint fixpoint semantics*. The intended joint models are defined to be the (minimal) joint fixpoints of the agent programs. We study computational properties of this new semantics showing that determining whether two (or more) logic programs have a joint fixpoint is NP complete. This remains true even for entirely positive logic programs. We also study the complexity of skeptical and credulous reasoning under the joint fixpoint semantics. The former is proven to be co-NP complete, while the latter is Σ_2^P complete. We show how the joint fixpoints of a set of logic programs can be computed as stable sets.

1 The Joint Fixpoint Semantics for Finding Compromises

Assume there are three agents, Mary, Larry, and Brenda, who discuss about dinner. Mary and Larry care much about food. Brenda is very tolerant about food. She is picky about drinks, however. Here are their respective requirements: **Mary:** I would like to have soup. I'd like to have either meat or fish this evening. I don't like potatoes. Spinach is okay. Carrots are okay, too (but I don't necessarily care for any of those). Concerning drinks, I have no real preference among beer, red wine, and white wine. **Larry:** Soup is fine (but not a must). However, if we have soup, I want to eat meat. Fish is okay. I'd like to have either spinach or potatoes. Carrots (in addition) are okay for me, if somebody wants them. Every drink (among beer, red or white wine) is fine. **Brenda:** Whatever you decide about food is okay for me. However, I care much about drinks. If we eat fish I insist on white wine, and if we eat meat, red wine is okay (but not a must).

Requests and consents of the above form can be expressed in logic programming with negation as failure (*not*), with an absurdity sign $\bot$ and with an additional modality *okay*(p), meaning that p is *tolerated*. Modal atoms *okay*(p)

can only appear in the head of a rule.[1] This means that p is not requested, but accepted if necessary in order to reach a compromise. A program written in this enriched language is referred to as a *Compromise Logic Program (COLP)*. The desires and consents of Mary, Larry, and Brenda are represented by the following COLPS P_m, P_l, and P_b, respectively:

P_m :

$$
\begin{aligned}
\bot &\leftarrow \texttt{potatoes} \\
okay(\texttt{spinach}) &\leftarrow \\
okay(\texttt{carrots}) &\leftarrow \\
\texttt{soup} &\leftarrow \\
\texttt{fish} &\leftarrow \mathit{not}\ \texttt{meat} \\
\texttt{meat} &\leftarrow \mathit{not}\ \texttt{fish} \\
okay(\texttt{redwine}) &\leftarrow \\
okay(\texttt{whitewine}) &\leftarrow \\
okay(\texttt{beer}) &\leftarrow
\end{aligned}
$$

P_l :

$$
\begin{aligned}
\texttt{potatoes} &\leftarrow \mathit{not}\ \texttt{spinach} \\
\texttt{spinach} &\leftarrow \mathit{not}\ \texttt{potatoes} \\
okay(\texttt{soup}) &\leftarrow \\
\texttt{meat} &\leftarrow \texttt{soup} \\
okay(\texttt{fish}) &\leftarrow \\
okay(\texttt{carrots}) &\leftarrow \\
okay(\texttt{redwine}) &\leftarrow \\
okay(\texttt{whitewine}) &\leftarrow \\
okay(\texttt{beer}) &\leftarrow
\end{aligned}
$$

P_b :

$$
\begin{aligned}
okay(\texttt{spinach}) &\leftarrow \\
okay(\texttt{carrots}) &\leftarrow \\
okay(\texttt{soup}) &\leftarrow \\
okay(\texttt{potatoes}) &\leftarrow \\
okay(\texttt{fish}) &\leftarrow \\
okay(\texttt{meat}) &\leftarrow \\
okay(\texttt{redwine}) &\leftarrow \texttt{meat} \\
\texttt{whitewine} &\leftarrow \texttt{fish} \\
okay(\texttt{beer}) &\leftarrow
\end{aligned}
$$

What we are looking for is a good semantics, which allows us to determine the intuitively intended models representing the acceptable compromises satisfying all requirements of the agents taking into account also their consents (i.e., the *okay* statements). To this aim, let us first more or less informally specify some desiderata of such a semantics.

Requirements:

1. Every intended model should be a model of each single agent's program (when *okay*-clauses are disregarded and $\bot$ is interpreted as *false*).
2. For each agent COLP P, each intended model M, and each atom $p \in M$, one of the two following conditions should hold: (a) p is supported in the classical sense by M in P, i.e., there is at least one rule of P with head p and body true in M; or (b) p is supported by some other agent, and $okay(p)$ is supported by M in P. (This is the case when the agent corresponding to P accepts p by compromise.)

[1] We make this restriction for simplicity here. Our semantics could be extended to programs containing *okay* literals in rule bodies, too.

3. M should be as small as possible, i.e., no unnecessary atoms should be contained in M.
4. For any agent program P, the body atoms of a clause of P whose head is $\bot$ cannot be simultaneously satisfied by M.

It is easy to see that these requirements are *not* fulfilled, if we consider any semantics that operates on the union of all agent programs (i.e., on the single program obtained by putting together all agent programs). In fact, by performing such a union, we would unite all *okay* statements and thus risk to be more liberal than intended. Any satisfactory approach to fulfill the above requirements must thus operate on the *set* of the agent programs and not on their union.

The first contribution of this paper is to present a semantics for "compromise logic programming" that satisfies all the requirements and appears to be extremely natural, intuitive, and clear-cut. This semantics is referred to as the *Joint Fixpoint Semantics (JFP Semantics)*. Interestingly, it completely relies on the well-known fixpoint semantics for logic programs with negation [14]. The JFP semantics, and, in particular, the new *okay* modality, is fully explained in terms of classical logic program constructs. We define a function σ mapping a COLP into a classical logic program as follows. (i) For each COLP P, we have: $\sigma(P) = \{\sigma(r) \mid r \text{ is a rule of } P\}$. (ii) Each classical rule r (i.e., rule in whose head neither *okay* nor $\bot$ appears) is invariant under σ, i.e., $\sigma(r) = r$. (iii) For each modal rule $r = okay(p) \leftarrow body$, we have: $\sigma(r) = p \leftarrow p, body$. (iv) For each rule $r = \bot \leftarrow body$ appearing in some agent program P_i, let abs_i be a new atom occurring in no other program P_j, $j \neq i$, and let $\sigma(r) = abs_i \leftarrow body$.

A *fixpoint* of a (classical) LP P is a supported model of P. A formal definition will be given in Section 2. Recall that each positive LP has a unique minimal fixpoint. A program with negation in rule bodies may have several minimal fixpoints. We denote by $FP(Q)$ the set of all fixpoints of a LP Q. If $T = \{Q_1, \ldots, Q_n\}$ is a set of (classical) LPs defined over the same set of atoms, then $JFP(T)$ denotes the set of *joint fixpoints* of $Q_1 \ldots, Q_n$:

$$JFP(T) = JFP(Q_1, Q_2, \ldots, Q_n) = FP(Q_1) \cap FP(Q_2) \cap \cdots \cap FP(Q_n).$$

By $MJFP(T)$ or $MJFP(Q_1, \ldots, Q_n)$ we denote the set of all set-minimal elements of $JFP(T)$. We are now ready for specifying the *joint fixpoint semantics for COLPs*. We do this, by assigning to each set $S = \{P_1, P_2 \ldots, P_n\}$ of COLPs a set of intended models $\mathcal{M}(S)$ by:

$$\mathcal{M}(S) = MJFP(\sigma(S)) = MJFP(\sigma(P_1), \sigma(P_2), \ldots, \sigma(P_n)).$$

The following proposition is easy to verify (we do not give a formal proof here).

Proposition 1. *The joint fixpoint semantics for COLPs satisfies all requirements* $1-4$.

To illustrate the joint fixpoint semantics, consider the programs $\sigma(P_m)$, $\sigma(P_l)$, and $\sigma(P_b)$ of our example programs:

$\sigma(P_m)$:

$$
\begin{array}{rcl}
abs_m & \leftarrow & \texttt{potatoes} \\
\texttt{spinach} & \leftarrow & \texttt{spinach} \\
\texttt{carrots} & \leftarrow & \texttt{carrots} \\
\texttt{soup} & \leftarrow & \\
\texttt{fish} & \leftarrow & \textit{not}\ \texttt{meat} \\
\texttt{meat} & \leftarrow & \textit{not}\ \texttt{fish} \\
\texttt{redwine} & \leftarrow & \texttt{redwine} \\
\texttt{whitewine} & \leftarrow & \texttt{whitewine} \\
\texttt{beer} & \leftarrow & \texttt{beer}
\end{array}
$$

$\sigma(P_l)$:

$$
\begin{array}{rcl}
\texttt{potatoes} & \leftarrow & \textit{not}\ \texttt{spinach} \\
\texttt{spinach} & \leftarrow & \textit{not}\ \texttt{potatoes} \\
\texttt{soup} & \leftarrow & \texttt{soup} \\
\texttt{meat} & \leftarrow & \texttt{soup} \\
\texttt{fish} & \leftarrow & \texttt{fish} \\
\texttt{carrots} & \leftarrow & \texttt{carrots} \\
\texttt{redwine} & \leftarrow & \texttt{redwine} \\
\texttt{whitewine} & \leftarrow & \texttt{whitewine} \\
\texttt{beer} & \leftarrow & \texttt{beer}
\end{array}
$$

$\sigma(P_b)$:

$$
\begin{array}{rcl}
\texttt{spinach} & \leftarrow & \texttt{spinach} \\
\texttt{carrots} & \leftarrow & \texttt{carrots} \\
\texttt{soup} & \leftarrow & \texttt{soup} \\
\texttt{potatoes} & \leftarrow & \texttt{potatoes} \\
\texttt{fish} & \leftarrow & \texttt{fish} \\
\texttt{meat} & \leftarrow & \texttt{meat} \\
\texttt{redwine} & \leftarrow & \texttt{redwine}, \texttt{meat} \\
\texttt{whitewine} & \leftarrow & \texttt{fish} \\
\texttt{beer} & \leftarrow & \texttt{beer}
\end{array}
$$

It is easily verifiable that $\sigma(P_m)$, $\sigma(P_l)$ and $\sigma(P_b)$ admit an unique minimal joint fixpoint that is $M = \{\texttt{soup}, \texttt{meat}, \texttt{spinach}\}$. Examples of joint fixpoints that are not minimal are $M_1 = \{\texttt{soup}, \texttt{meat}, \texttt{spinach}, \texttt{carrots}\}$ and $M_1 = \{\texttt{soup}, \texttt{meat}, \texttt{spinach}, \texttt{redwine}\}$. Note that a set of COLPs can have multiple intended models. For example, if we replace the rule $\bot \leftarrow \texttt{potatoes}$ by the rule $\mathit{okay}(\texttt{potatoes}) \leftarrow$ in the program P_m we obtain that the programs $\sigma(P_m)$, $\sigma(P_l)$ and $\sigma(P_b)$ admit as minimal joint fixpoint also the model $M' = \{\texttt{soup}, \texttt{meat}, \texttt{potatoes}\}$ in addition to M.

We have thus introduced a completely new semantics for describing compromises of agents who declare their requirements and their consents. This semantics is based on a new use of the classical machinery of fixpoints of logic programs, in particular, all minimal joint fixpoints of the logic programs associated to the given COLPs. Note that our translation from COLPs to classical programs provides a new meaning to clauses of the form $p \leftarrow p$. In the classical single-program fixpoint semantics, such clauses have the somewhat questionable meaning "p can be opted to be part of a fixpoint at any time". In our multiagent context, such clauses correspond to modal atoms $\mathit{okay}(p)$ and have the following precise meaning: "if p is required by *another* agent, then let it be".

The computationally interesting tasks associated with the joint fixpoint semantics are the following: (1) **Joint Fixpoint Existence**: Determining whether a set of LPs has a joint fixpoint (of course, there is a minimal JFP iff there is a JFP). This corresponds to determining whether a set of agents can reach a

compromise at all. (2) **Skeptical reasoning under the JFP semantics**: This means determining whether some atom p occurs in all minimal JFPs of some logic programs $P_1, \ldots, P_n$. Note that this is equivalent to determining whether p occurs in all JFPs of $P_1, \ldots, P_n$. In this case, any compromise will force all agents to adopt p. This is of course an interesting information worth to be known. (3) **Credulous Reasoning:** This means, determining whether some atom p occurs in *at least one* minimal JFP. In practice this means that a compromise containing p may be chosen.

In Section 4 we study the complexity of these three problems. In particular, we show that: (i) JFP existence is NP complete even for pairs P_1, P_2 of *purely positive* programs (in which neither the symbol *not*, nor the absurdity symbol appear). This is rather astonishing, because each positive program has a unique least fixpoint, and one could have thought that a joint fixpoint could be constructed from least fixpoints of P_1 and P_2; (ii) Skeptical JFP reasoning is co-NP complete and thus exactly as hard as inferencing in classical propositional logic, or as reasoning under the stable model semantics [9, 20, 21, 19]; (iii) Credulous JFP reasoning is Σ_2^p-complete, and thus exactly as hard as credulous reasoning in default logic [18, 8, 11], or as circumscriptive reasoning [16, 4, 11], or as *disjunctive* logic programs under the stable model semantics [5]. In Section 5 we construct a polynomial-time translation between the JFP semantics and the stable model semantics for LPs with negation [9]. The advantage of such a translation is that existing engines [7, 17] (that are rather efficient in practice) for computing stable models can be used to compute joint fixpoints. In particular, the tasks *JFP existence* and *skeptical JFP reasoning* can be easily translated to analogous LP tasks according to the stable model semantics. This enables the construction of simple frontends for JFP reasoning to systems such as S-models [17], the dlv system [7], or others. Before proceeding with our technical exposition, let us discuss related work. To the best of our knowledge, we are not aware of similar approaches. The work in the area of Belief-Desire-Intention is mainly based on various modal logics [24 26]. Closest to our work is perhaps [22]. This approach is based on logic programs too and considers diagnostic agents that need to reach a common diagnosis. So the problem is similar to our setting but their methods to solve it are not. There is also the CaseLP approach in [15] based on logic programs, but the authors do not consider the problem to compute common conclusions between the agents. Various methods for giving semantics to logic programs with *conflicting rules* have been defined in the literature (e.g. Ordered Logic Programming [2, 3], the PARK model [10], or Courteous Logic Programming for prioritized conflict handling [12]). An interesting extension of logic programming to provide multi-agent functionality is presented in [13]. This model is, however, very different from ours and has completely different aims. Its goal is not to reach common conclusions or compromises, but to achieve a "thinking component" of an agent via a proof procedure the combines abductive backward reasoning with a forward reasoning method that uses constraint checking methods in the style of Constraint Logic Programming. An agent can observe changes as inputs and react to them under time

resource bounds, using an "agent cycle" that alternates between thinking operations, choices, and actions. In summary, this proof-theoretic model of dynamic agent interaction with time parameters cannot be reasonably compared to our method of model-theoretically defining the concept of a compromise of agent desires that are (statically) defined through a set of logic programs. Finally, there is the IMPACT project [23], a multiagent framework the underlying semantics of which is also based on logic programs. Although the authors do not consider explicitly the problem of reaching common conclusions, it seems that their use of (flat) modalities might be used to encode some of the examples considered in this paper. But in all the above cases, our use of a joint fixpoint of a set of logic programs is new and has not been considered before.

For space limitations proofs of theorems are not given here. They can be found in the full version of this paper [1].

2 Preliminaries on Logic Programming

A propositional logic program $\mathcal{P}$ is defined on a finite set of propositional variables $Var(\mathcal{P})$. An *atom* or *positive literal* of $\mathcal{P}$ is an element $a \in Var(\mathcal{P})$; a *negative literal* is the negation *not* a of an atom. A *program clause* or *rule* r is

$$a \leftarrow b_1 \wedge \cdots \wedge b_k \wedge not\ b_{k+1} \wedge \cdots \wedge not\ b_m \qquad m \geq 0.$$

where $a, b_1, \cdots, b_k$ are positive literals and *not* $b_{k+1}, \cdots,$ *not* b_m are negative literals. a is called the *head* of r, while the conjunction $b_1 \wedge \cdots \wedge b_k \wedge not\ b_{k+1} \wedge \cdots \wedge not\ b_m$ is its *body*. A (propositional) *logic program* $\mathcal{P}$ consists of a finite set of program clauses whose propositional variables are all in $Var(\mathcal{P})$. (Note, however, that $Var(\mathcal{P})$ may contain atoms that do not occur in $\mathcal{P}$). We denote by $Var^*(\mathcal{P})$ the set of atoms of $Var(\mathcal{P})$ appearing in $\mathcal{P}$. A logic program is *positive* if no negative literal occurs in it. An *(Herbrand) interpretation* for a program $\mathcal{P}$ is a subset of $Var(\mathcal{P})$. A positive literal a (resp. a negative literal *not* a) is *true* w.r.t. an interpretation I if $a \in I$ (resp. $a \notin I$); otherwise it is *false*. A rule is *satisfied* (or is *true*) w.r.t. I if its head is true or its body is false w.r.t. I. An interpretation I is a *(Herbrand) model* of a program $\mathcal{P}$ if it satisfies all rules in $\mathcal{P}$. For each program $\mathcal{P}$, the *immediate consequence operator* $T_{\mathcal{P}}$ is a function from $2^{Var(\mathcal{P})}$ to $2^{Var(\mathcal{P})}$ defined as follows. For each interpretation $I \subseteq Var(\mathcal{P})$, $T_{\mathcal{P}}(I)$ consists of the set of all heads of rules in $\mathcal{P}$ whose bodies evaluate to true in I. Note that $T_{\mathcal{P}}$ is well-defined also for programs with negations in rule bodies. An interpretation I is a *fixpoint* of a logic program $\mathcal{P}$ if I is a fixpoint of the associated transformation $T_{\mathcal{P}}$, i.e., if $T_{\mathcal{P}}(I) = I$. Note that each fixpoint of $\mathcal{P}$ is also a model of $\mathcal{P}$, but the converse does not hold in general. For example the program consisting of the single rule $q \leftarrow p$ has as unique fixpoint the empty set; however, the interpretation $M = \{p, q\}$ is a model of $\mathcal{P}$. The set of all fixpoints of $\mathcal{P}$ is denoted by $FP(\mathcal{P})$. Let I be an interpretation of $\mathcal{P}$ and let $a \in Var(\mathcal{P})$ be an atom. We say that a is *supported* by I (in $\mathcal{P}$) if there is a rule of $\mathcal{P}$ with head a whose body evaluates to true in I, i.e., if $a \in T_{\mathcal{P}}(I)$. From the definition of fixpoint it immediately follows that an interpretation I of $\mathcal{P}$ is a fixpoint of $\mathcal{P}$

iff I coincides with the set of all atoms supported by I. For any interpretation $I \subseteq Var(\mathcal{P})$, we define $T_{\mathcal{P}}^0(I) = I$ and for all $i \geq 0$, $T_{\mathcal{P}}^{i+1}(I) = T_{\mathcal{P}}(T_{\mathcal{P}}^i(I))$. If $\mathcal{P}$ is a positive program, then $T_{\mathcal{P}}$ is monotonic and thus has a least fixpoint $lfp(\mathcal{P}) = T_{\mathcal{P}}^{\infty}(\emptyset)$. This least fixpoint coincides with the least Herbrand model $lm(\mathcal{P})$ of $\mathcal{P}$, i.e., $lm(\mathcal{P}) = lfp(\mathcal{P})$. For non-positive programs $\mathcal{P}$, $T_{\mathcal{P}}$ is in general not monotonic, and $\mathcal{P}$ does not necessarily have a least fixpoint (it may even have no fixpoint at all). It was shown in [14] that it is NP complete to determine whether a non-positive logic program has a fixpoint.

Now we recall the notion of stable models for propositional logic programs and we report some results from [6] that we shall use in the following. Let $\mathcal{P}$ be a logic program and $I \subseteq Var(\mathcal{P})$ be an interpretation. The *Gelfond-Lifschitz transformation* (or simply *GL-transformation*) of $\mathcal{P}$ w.r.t. I, denoted by $\mathcal{P}^I$ is the program obtained by $\mathcal{P}$ by removing all rules containing a negative literal *not* b in the body such that $b \in I$, and by removing all negative literals from the remaining rules.

Definition 1 ([9]). *Given a logic program $\mathcal{P}$ and an interpretation $M \subseteq Var(\mathcal{P})$, M is a* stable model *of $\mathcal{P}$ if $M = T_{\mathcal{P}^M}^{\infty}(\emptyset)$.*

A logic program $\mathcal{P}$ admits in general a number (possibly zero) of stable models. We denote by $SM(\mathcal{P})$ the set of all stable models of the program $\mathcal{P}$.

Definition 2 ([6]). *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be programs. We say that $\mathcal{P}_2$* potentially uses *$\mathcal{P}_1$ (denoted $\mathcal{P}_2 \rhd \mathcal{P}_1$) if each predicate that occurs in some rule head of $\mathcal{P}_2$ does not occur (positively or negatively) in $\mathcal{P}_1$.*

Given a set of atoms M, the *program of* M is the set of rules $\{a \leftarrow \mid a \in M\}$. With a little abuse of notation, when the context is clear, we denote the program of a set of atoms M by the same symbol M.

Proposition 2 ([6]). *Let $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2$ be a program such that $\mathcal{P}_2 \rhd \mathcal{P}_1$. Then: $SM(\mathcal{P}) = \bigcup_{M \in SM(\mathcal{P}_1)} SM(M \cup \mathcal{P}_2)$.*

Proposition 3 ([6]). *Let $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2$ be a program such that $Var^*(\mathcal{P}_1) \cap Var^*(\mathcal{P}_2) = \emptyset$.[2] Then: $SM(\mathcal{P}) = \bigcup_{M_1 \in SM(\mathcal{P}_1), M_2 \in SM(\mathcal{P}_2)} \{M_1 \cup M_2\}$.*

3 Joint Fixpoints

In this section we introduce the Joint Fixpoint Semantics for logic programs.

Let $\mathcal{P}_1, \mathcal{P}_2, \ldots \mathcal{P}_n$ be logic programs such that $Var(\mathcal{P}_1) = Var(\mathcal{P}_2) = \cdots = Var(\mathcal{P}_n)$. We define the set $JFP(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n)$ of *joint fixpoints* by:

$$JFP(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n) = FP(\mathcal{P}_1) \cap FP(\mathcal{P}_2) \cap \cdots \cap FP(\mathcal{P}_n).$$

In words, $JFP(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n)$ consists of all common fixpoints to the programs $\mathcal{P}_1, \ldots, \mathcal{P}_n$. Moreover, we define the set $MJFP(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ of *minimal joint fixpoint* as:

[2] Recall that $Var^*(\mathcal{P})$ denotes the set of atoms actually appearing in $\mathcal{P}$.

$$MJFP(\mathcal{P}_1,\ldots,\mathcal{P}_n) = \{F \in JFP(\mathcal{P}_1,\ldots,\mathcal{P}_n) \mid \nexists F' \in JFP(\mathcal{P}_1,\ldots,\mathcal{P}_n) \wedge F' \subset F\}.$$

$MJFP(\mathcal{P}_1,\ldots,\mathcal{P}_n)$ consists of all minimal common fixpoints to the programs $\mathcal{P}_1,\ldots,\mathcal{P}_n$.

Since, as mentioned, it is NP complete to determine whether a *single* non-positive program has a fixpoint, determining whether a set of programs containing at least one non-positive program has a joint fixpoint is trivially NP hard. Moreover, since this problem is easily seen to be in NP, it is NP complete. In this paper we are also interested in joint fixpoints of *positive* programs. In particular, we will investigate the issue whether a set of positive programs has a minimal JFP and we will study different forms of *reasoning* with joint fixpoints. The following example shows that a set of positive logic programs may have zero, one, or more joint fixpoints.

Example 1.

- If $\mathcal{P}_1 = \{p \leftarrow\}$ and $\mathcal{P}_2 = \{q \leftarrow\}$, then $JFP(\mathcal{P}_1,\mathcal{P}_2) = MJFP(\mathcal{P}_1,\mathcal{P}_2) = \emptyset$.
- If $\mathcal{P}_1 = \{p \leftarrow q\}$ and $\mathcal{P}_2 = \{p \leftarrow s\}$, then $JFP(\mathcal{P}_1,\mathcal{P}_2) = MJFP(\mathcal{P}_1,\mathcal{P}_2) = \{\emptyset\}$.
- If $\mathcal{P}_1 = \{p \leftarrow\}$ and $\mathcal{P}_2 = \{p \leftarrow p\}$, then $JFP(\mathcal{P}_1,\mathcal{P}_2) = MJFP(\mathcal{P}_1,\mathcal{P}_2) = \{\{p\}\}$.
- If $\mathcal{P}_1 = \{p \leftarrow p,\ q \leftarrow q\}$ and $\mathcal{P}_2 = \{p \leftarrow q,\ q \leftarrow p\}$, then $JFP(\mathcal{P}_1,\mathcal{P}_2) = \{\emptyset, \{p,q\}\}$ and $MJFP(\mathcal{P}_1,\mathcal{P}_2) = \{\emptyset\}$.

We will also consider *credulous and skeptical reasoning* under joint fixpoints. Let $S = \{\mathcal{P}_1,\ldots,\mathcal{P}_n\}$ be a set of logic programs over the same set of propositional variables Var. Let p be an atom in Var.

- p is a *credulous MJFP-consequence* of S if for some minimal joint fixpoint $I \in MJFP(\mathcal{P}_1,\ldots,\mathcal{P}_n)$ it holds that $p \in I$.
- p is a *skeptical MJFP-consequence* of S if for all minimal joint fixpoints $I \in MJFP(\mathcal{P}_1,\ldots,\mathcal{P}_n)$ it holds that $p \in I$.

We define the following decision problems:
PROBLEM JFP (JFP existence):

Instance: A set of positive logic programs $\mathcal{P}_1,\ldots,\mathcal{P}_n$ defined over the same set of propositional variables.
Question: Is $JFP(\mathcal{P}_1,\ldots\mathcal{P}_n) \neq \emptyset$, i.e., do the programs $\mathcal{P}_1,\ldots,\mathcal{P}_n$ have a joint fixpoint?

The problem JFP_2 is the restriction of JFP to instances consisting of two positive programs:
PROBLEM JFP_2 (JFP existence restricted to the case of two programs):

Instance: A pair of positive logic programs $\mathcal{P}_1$ and $\mathcal{P}_2$.

Question: Is $JFP(\mathcal{P}_1, \mathcal{P}_2) \neq \emptyset$, i.e., do the programs $\mathcal{P}_1$ and $\mathcal{P}_2$ have a joint fixpoint?

PROBLEM MJFP^s (skeptical reasoning under the JFS semantics):

Instance A set of positive logic programs $S = \{\mathcal{P}_1, ..., \mathcal{P}_n\}$ defined over the same set of propositional variables Var and an atom $p \in Var$.
Question Is p a skeptical MJFP-consequence of S?

PROBLEM MJFP^c (credulous reasoning under the JFS semantics):

Instance A set of positive logic programs $S = \{\mathcal{P}_1, ..., \mathcal{P}_n\}$ defined over the same set Var of propositional variables and and an atom $p \in Var$.
Question Is p a credulous MJFP-consequence of S?

Also in this case, we define the restrictions of MJFP^s and MJFP^c to the case in which S contains only two programs. We denote such decision problems by MJFP$^s_2$ and MJFP$^c_2$, respectively.

4 Complexity Results

Theorem 1. *The problems* JFP *and* JFP$_2$ *are NP complete.*

Theorem 2. *The problems* MJFP^s *and* MJFP$^s_2$ *are co-NP complete.*

Now we analyze the complexity of the problems MJFP^c and MJFP$^c_2$. First we give some preliminary definition and results.

A *positive propositional disjunctive logic program* (DL^+*-program*) is a positive propositional theory in DNF. We denote the i-th rule of a DL^+-program consisting of $t > 0$ rules by: $h^i_1 \vee \cdots \vee h^i_{n_i} \leftarrow body(r_i)$, where $n_i > 0$, $body(r_i)$ denotes a (possibly empty) conjunction of positive literals, and r_i is a label not occurring in $Var(\mathcal{P})$ identifying the rule i, for each $1 \leq i \leq t$.

The models $M(\mathcal{P})$ of the DL^+-program $\mathcal{P}$ precisely coincide to classical models of the program seen as positive propositional theory in DNF. The same happens for the minimal models $MM(\mathcal{P})$.

Definition 3. *Let $\mathcal{P}$ be a DL^+-program consisting of $t > 0$ rules. $R_\mathcal{P}$ is a (disjunction-free) program over the set of propositional variables $Var(\mathcal{P}) \cup \{r_1, ..., r_t\}$ consisting of the following set rules: $R_\mathcal{P} = \{r_i \leftarrow h^i_j \mid 1 \leq i \leq t \wedge 1 \leq j \leq n_i\}$. Moreover, we define the DL^+-program $\mathcal{P}^* = \mathcal{P} \cup R_\mathcal{P}$.*

Lemma 1. *Let $\mathcal{P}$ be a DL^+-program. Then: $MM(\mathcal{P}^*) = \bigcup_{M \in MM(\mathcal{P})} MM(M \cup R_\mathcal{P})$.*

We recall that the minimal model semantics assigns to $\mathcal{P}$ the set $MM(\mathcal{P})$ of minimal models of $\mathcal{P}$. A propositional formula ϕ is a *credulous consequence* under the minimal model semantics of $\mathcal{P}$ if for some $M \in MM(\mathcal{P})$ it holds that $M \models \phi$. Observe that the problem of deciding whether a propositional formula is a credulous consequence under the minimal model semantics of a positive disjunctive program is Σ^P_2-complete [5, 6].

Lemma 2. *Given a DL^+-program $\mathcal{P}$ and an atom $p \in Var(\mathcal{P})$, p is a credulous consequence under the minimal models semantics of $\mathcal{P}$ if and only if it is a credulous consequence under the minimal models semantics of $\mathcal{P}^*$.*

Definition 4. *Let $\mathcal{P}$ be a DL^+-program consisting of $t > 0$ rules. We define the set $JUST^r(\mathcal{P}^*)$ as the set of models M of $\mathcal{P}^*$ such that each atom in $M \cap \{r_1, ..., r_t\}$ is supported by M (in $\mathcal{P}^*$). $JUST^r(\mathcal{P}^*)$ is said the set of* rule-justified models *(or simply* r-justified models*)* of $\mathcal{P}^*$.

Moreover, we define the set $MJUST^r(\mathcal{P}^*)$ of minimal r-justified models *of $\mathcal{P}^*$ as: $MJUST^r(\mathcal{P}^*) = \{M \in JUST^r(\mathcal{P}^*) \mid \nexists M' \in JUST^r(\mathcal{P}^*) \wedge M' \subset M\}$.*

Lemma 3. *Let $\mathcal{P}$ be a DL^+-program. Then: $MM(\mathcal{P}^*) = MJUST^r(\mathcal{P}^*)$.*

Definition 5. *Let $\mathcal{P}$ be a DL^+-program consisting of $t > 0$ rules. We define the two programs $\mathcal{P}_h$ and $\mathcal{P}_b$ associated to the program $\mathcal{P}$ in the following way:*
The program $\mathcal{P}_h$ is the union of the sets of rules S_h^1 and S_h^2 defined as follows:

$$S_h^1 = \{r_i \leftarrow h_j^i \mid 1 \leq i \leq t \wedge 1 \leq j \leq n_i\}$$
$$S_h^2 = \{x \leftarrow x \mid x \in Var(\mathcal{P})\}$$

The program $\mathcal{P}_b$ is the union of the sets of rules S_b^1 and S_b^2 defined as follows:

$$S_b^1 = \{r_i \leftarrow body(r_i) \mid 1 \leq i \leq t\}$$
$$S_b^2 = \{x \leftarrow x \mid x \in Var(\mathcal{P}) \cup \{r_1, ..., r_t\}\}.$$

Lemma 4. *Let $\mathcal{P}$ be a DL^+-program. Then: $JFP(\mathcal{P}_h, \mathcal{P}_b) = JUST^r(\mathcal{P}^*)$.*

Lemma 5. *Given a set of positive logic programs S over the same set of propositional variables Var, and a set $F \subseteq Var$, deciding whether F is a minimal joint fixpoint of S is in co-NP.*

Theorem 3. *The problems* MJFP^c *and* MJFP$_2^c$ *are Σ_2^P-complete.*

5 Joint Fixpoints and Stable Models

In this section we give the translation from Logic Programming under the Joint Fixpoint Semantics to Logic Programming under Stable Model Semantics. First we need some preliminary definitions and results.

Definition 6. *Let $\mathcal{P}$ be a program and let M be a set of atoms in $Var(\mathcal{P})$. We denote by $[M]_\mathcal{P}$ the set $\{a_\mathcal{P} \mid a \in M\} \cup \{a'_\mathcal{P} \mid a \in Var(\mathcal{P}) \setminus M\} \cup \{sa_\mathcal{P} \mid a \in M\}$.*

Definition 7. *Let $\mathcal{P}$ be a positive program. We define the program $\Gamma(\mathcal{P})$ over the set of atoms $Var(\Gamma(\mathcal{P})) = \{a_\mathcal{P} \mid a \in Var(\mathcal{P})\} \cup \{a'_\mathcal{P} \mid a \in Var(\mathcal{P})\} \cup \{sa_\mathcal{P} \mid a \in Var(\mathcal{P})\} \cup \{fail_\mathcal{P}\}$ as the union of the sets of rules S_1, S_2 and S_3, defined as follows:*

$$S_1 = \{a_\mathcal{P} \leftarrow not\ a'_\mathcal{P} \mid a \in Var(\mathcal{P})\} \cup \{a'_\mathcal{P} \leftarrow not\ a_\mathcal{P} \mid a \in Var(\mathcal{P})\}$$

$$S_2 = \{sa_\mathcal{P} \leftarrow b_\mathcal{P}^1, ..., b_\mathcal{P}^n \mid a \leftarrow b_1, \ldots b_n \in \mathcal{P}\}$$

$$S_3 = \{fail_\mathcal{P} \leftarrow not\ fail_\mathcal{P}, sa_\mathcal{P}, not\ a_\mathcal{P} \mid a \in Var(\mathcal{P})\} \cup$$
$$\{fail_\mathcal{P} \leftarrow not\ fail_\mathcal{P}, a_\mathcal{P}, not\ sa_\mathcal{P} \mid a \in Var(\mathcal{P})\}.$$

Lemma 6. *Let $\mathcal{P}$ be a program. Then: $SM(\Gamma(\mathcal{P})) = \bigcup_{F \in FP(\mathcal{P})} \{[F]_{\mathcal{P}}\}$.*

An immediate consequence of the above lemma is that there is a one-to-one correspondence between the set of fixpoints of a given program $\mathcal{P}$ and the set $SM(\Gamma(\mathcal{P}))$ of stable models of the program $\Gamma(\mathcal{P})$.

Now suppose we have a set of positive programs $\mathcal{P}_1, \ldots, \mathcal{P}_n$ over the same set of propositional variables. We find a program $J(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ associated to the set of programs $\mathcal{P}_1, \ldots, \mathcal{P}_n$ such that the stable models of $J(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ correspond to the joint fixpoints of $\mathcal{P}_1, \ldots, \mathcal{P}_n$. $J(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ is constructed by performing the union of all the programs $\Gamma(\mathcal{P}_i)$, for $1 \leq i \leq n$, with another program $C(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ that we next define. Informally, under stable model semantics, rules of programs $\Gamma(\mathcal{P}_1)$, $\Gamma(\mathcal{P}_2)$, ... , $\Gamma(\mathcal{P}_n)$ have the effect of generating all the fixpoints of $\mathcal{P}_1$, $\mathcal{P}_2$, ... , $\mathcal{P}_n$, respectively, while rules of $C(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ select among these all fixpoints that are simultaneously fixpoints of $\mathcal{P}_1$, $\mathcal{P}_2$, ... , $\mathcal{P}_n$.

Definition 8. *Given a set of positive programs $\mathcal{P}_1, \ldots, \mathcal{P}_n$ over the same set of atomic propositions Var, $C(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ is the program over $Var' = \bigcup_{1 \leq i \leq n} \{a_{\mathcal{P}_i} \mid a \in Var\} \cup \{fail\}$ defined as follows:*

$$C(\mathcal{P}_1, \ldots, \mathcal{P}_n) = \{fail \leftarrow not\ fail, a_{\mathcal{P}_i}, not\ a_{\mathcal{P}_j} \mid 1 \leq i \neq j \leq n\}.$$

Moreover, the program $J(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ over $\bigcup_{1 \leq i \leq n} Var(\Gamma(\mathcal{P}_i)) \cup \{fail\}$ is defined as:

$$J(\mathcal{P}_1, \ldots, \mathcal{P}_n) = \Gamma(\mathcal{P}_1) \cup \cdots \cup \Gamma(\mathcal{P}_n) \cup C(\mathcal{P}_1, \ldots, \mathcal{P}_n).$$

The next theorem states that there is a one-to-one correspondence between the set of joint fixpoints of the programs $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and the set of stable models of the program $J(\mathcal{P}_1, \ldots, \mathcal{P}_n)$.

Theorem 4. *Let $\mathcal{P}_1, \ldots, \mathcal{P}_n$ be positive logic programs over the same set of atomic propositions Var. Then:*

$$SM(J(\mathcal{P}_1, \ldots, \mathcal{P}_n)) = \bigcup_{F \in JFP(\mathcal{P}_1, \ldots, \mathcal{P}_n)} \{\cup_{1 \leq i \leq n} [F]_{\mathcal{P}_i}\}.$$

where $JFP(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ is the set of the joint fixpoints of $\mathcal{P}_1, \ldots, \mathcal{P}_n$.

6 Conclusion and Future Work

In this paper we have introduced a new model-theoretic semantics for defining compromises among desires and consents of agents represented by logic programs. Rather than joining the theories of different agents and considering models or fixpoints of a single joint logic program (possibly incorporating modalities), we advocated that the right approach is most likely to consider joint fixpoints of separate logic programs. To our best knowledge, the idea of using joint fixed points of logic programs is new and has never been explored by others. We think that this is a quite appealing idea, which uses existing concepts and machinery

in a diverse rendering. The effectiveness of our method was demonstrated on two small nontrivial examples. In this context, we also described a new way of specifying requests and consents of agents by logic programs. A novel feature is the *okay* modality and its translation into a self-implication of an atom in a classical logic program. Our new semantics for describing requests and consents of multiple agents naturally induced us to study the computational properties of reasoning with joint fixed points and with minimal joint fixed points. We proved the surprising result that determining whether two plain positive propositional logic programs have a joint fixpoint is already NP complete. Translated into our agent-compromise framework this means that determining whether there exists a compromise between two agents whose requests and consents are formulated in the simplest possible rule-based language (just definite propositional Horn clauses, without negation or disjunction or similar constructs) is a hard problem. For those who agree that our semantics can faithfully describe standpoints of agents, this NP hardness result says something about determining compromises in the real world. We also analyzed the complexity of credulous and skeptical reasoning under the minimal joint fixpoint semantics. We think that our complexity studies and results are of independent interest, whether one agrees with our interpretation of joint fixpoints as compromises or not.

Acknowledgments

Research supported by EU project WASP (Working Group on Answer Set Semantics) IST-2001-37004. We are grateful to Jürgen Dix anfd Thomas Eiter for helping us to find related work.

References

1. Buccafurri, F., Gottlob, G., Multiagent Compromises, Joint Fixpoints and Stable Models, *Computational Logic: From Logic Programming into the Future (In honour of Bob Kowalsky), Ed. A. Kakas and F. Sadri, Springer Verlag* 2002.
2. Buccafurri, F., Leone, N., Rullo, P.: Stable Models and their Computation for Logic Programming with Inheritance and True Negation. Journal of Logic Programming **27**(1), Elsevier Science (1996) 5–43
3. Buccafurri, F., Leone, N., Rullo, P.: Semantics and Espressiveness of Disjunctive Ordered Logic. Annals of Mathematics and Artificial Intelligence Journal. J.C. Balzer AC, Science Publisher **25** (1999) 311–337
4. Eiter, T., Gottlob, G.: Propositional Circumscription and Extended Closed-World Reasoning are Π_2^p-Complete. Theoretical Computer Science **114**(2) (1993) 231–245
5. Eiter, T., Gottlob, G.: On the Computational Cost of Disjunctive Logic Programming: Propositional Case. Annals of Mathematics and Artificial Intelligence. **15**(3-4) (1995) 289–323
6. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive Datalog. ACM Transactions on Database Systems **22**(3) (1997) 315–363
7. Eiter, T., Leone, N., Mateis, C., Pfeifer, G., Scarcello, F.: A Deductive System for Non-Monotonic Reasoning. Proc. of the 4th Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR '97). Lecture Notes in Computer Science, Vol. 1265. Springer-Verlag, Dagstuhl, Germany (1997) 364–375

8. Eiter, T., Gottlob, G., Cadoli, M.: Default Logic as a Query Language. Transactions on Knowledge Data Engineering **9**(3) (1997) 448–463
9. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. Proc. of the 5th International Conference on Logic Programming. MIT Press, Cambridge, (1988) 1070–1080
10. Gottlob, G., Moerkotte, G., Subrahmanian, V.S.: The PARK Semantics for Active Rules. Proc. of the Int. Conf. on Extending Database Technology, EDBT'96, Lecture Notes in Computer Science. Springer Verlag, (1996) 35–55
11. Gottlob, G.: Complexity Results for Nonmonotonic Logics. Journal of Logic and Computation **2**(3) (1992) 397–425
12. Grosof, B.: Prioritized Conflict Handling for Logic Programs. Proc. of the Int. Logic Programming Symposium, ILPS'97, MIT Press, Cambridge, (1997) 197–211.
13. Kowalski, R., and Sadri, F.: From LP Towards Multi-Agent Systems. Annals of Mathematics and Artificial Intelligence **25**(3-4) (1999) 391–419
14. Kolaitis, P.G., Papadimitriou, C.H.: Why not Negation by Fixpoint? Journal of Computer and System Sciences **43**(1) (1991) 125–144
15. Martelli, M., Mascardi V., Zini, F.: Towards Multi-Agent Software Prototyping. Proc. of The Third International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 98), London, UK, (1998) 331–354
16. McCarthy, J.: Circumscription - a Form of Nonmonotonic Reasoning, Artificial Intelligence **13** (1980) 27–39
17. Niemelä, I., Simons, P.: Smodels - an Implementation of the Stable Model and Well-founded Semantics for Normal Logic Programs. Proc. of the 4th Int. Conf. on Logic Programming and Non-Monotonic Reasoning (LPNMR '97), Lecture Notes in Computer Science, Vol. 1265. Springer-Verlag, Dagstuhl, Germany (1997) 420–429
18. Reiter, R.: A Logic for Default Reasoning. Artificial Intelligence **13** (1980) 81–132
19. Saccá, D.: The Expressive Power of Stable Models for Datalog Queries with Negation. Proc. of the ILPS'93 Workshop on Structural Complexity and Recursion-Theoretic Methods in Logic Programming, Washington D.C., USA (1993) 150–162
20. Schlipf, J.S.: The Expressive Powers of Logic Programming Semantics, Proc. of the ACM Symposium on Principles of Database Systems (1990) 196–204
21. Schlipf, J.S.: A Survey of Complexity and Undecidability Results in Logic Programming. Proc. of the ILPS'93 Workshop on Structural Complexity and Recursion-Theoretic Methods in Logic Programming, Washington D.C., USA (1993) 93–102
22. Schroeder, M., De Almeida Mora, I., and Pereira L.M.: A Deliberative and Reactive Diagnosis Agent based on Logic Programming. Intelligent Agents III: Lecture Notes in Artificial Intelligence **1193**, Springer-Verlag, J.P. Muller, M.J. Wooldridge and N. Jennings ed., (1997) 293–307
23. Subrahmanian V.S., Bonatti P., Dix, J., and Eiter T., Kraus S., Özcan, F., and Ross, R.: Heterogenous Active Agents. MIT-Press (2000)
24. Wooldridge, M., Jennings, N.R.: Formalizing the Cooperative Problem Solving Process. Readings in Agents, M. Huhns and M. Singh ed., Morgan Kaufmann (1997) 430–440
25. Wooldridge, M., Jennings, N.R.: Agent Theories, Architectures and Languages: A survey. Intelligent Agents, M. J. Wooldridge and N. R. Jennings ed., Lecture Notes in Artificial Intelligence, Springer-Verlag **890** (1995) 1–39.
26. Wooldridge, M., Jennings, N.R.: Intelligent Agents: Theory and Practice. Knowledge Engineering Reviews **10**(2) 1995