

Social Behavior of Agents and Stable models

Francesco Buccafurri and Gianluca Caminiti

DIMET, Università degli Studi Mediterranea di Reggio Calabria,
via Graziella loc. Feo di Vito, I-89060 Reggio Calabria, Italy,
{bucca, gianluca.caminiti}@unirc.it

Abstract. As in human world many of our goals could not be achieved without interacting with other people, in case many agents are part of the same environment one agent should be aware that he is not alone and he cannot assume other agents sharing his own goals. Moreover, he may be required to interact with other agents and to reason about their mental state in order to find out potential friends to join with (or opponents to fight against). In this paper we focus on a language derived from logic programming which both supports the representation of mental states of agent communities and provides each agent with the capability of reasoning about other agents' mental states and acting accordingly. The proposed semantics is shown to be translatable into stable model semantics of logic programs with aggregates.

1 Introduction

Beside *autonomy*, agents [16, 15] may be required to have *social ability*, which is the capability of interacting with other self-interested agents and, as a consequence, producing *beliefs*, *desires* and *intentions* (BDI) [2, 3] which may be dependent on such interactions.

Social ability means not only using a common language for agent communication. In this respect, KQML [13] and FIPA ACL [7], both based on the *speech act theory* by Cohen and Levesque [6], represent the main efforts done in the last years. Another important issue is reasoning about the content of such a communication [15, 14, 11]. In this paper we focus on a language derived from logic programming which both supports the representation of mental states of agent communities and provides each agent with the capability of reasoning about other agents' mental states and acting accordingly.

Consider the following example: There are four agents which have been invited to the same wedding party. Some agents are less autonomous than the others, i.e. they may decide either to join the party or not to go at all, possibly depending on the other agents' choice. Moreover some agents may tolerate some options. These are the desires of the agents:

Agent₁ will go to the party only if at least the half of the total number of agents (not including himself) goes there.

Agent₂ possibly does not go to the party, but he tolerates such an option. In case he goes, then he possibly drives the car.

Agent₃ would like to join the party together with **Agent**₂, but he is not so much safe with **Agent**₂'s driving skill. Thus he decides to go to the party only if **Agent**₂ both goes there and does not want to drive the car.

Agent₄ does not go to the party.

It is possible to represent the above desires using logic programming with negation as failure (*not*) where each agent is represented by a single program and requested/desired items (representing the mental state of the agent) are modelled as atoms occurring inside rule heads. In particular, mandatory items are modelled as facts. Moreover, it is possible to represent tolerated items, i.e. items which are not requested, but possibly accepted. To this aim we use the predicate *okay()*, previously introduced in [5].

However, representing the requests/acceptances of single agents in a community is not enough. A social language should provide also a machinery to handle compromises among those agents. Thus, we introduce a new construct providing one agent with the ability to reason about other agents' mental state and then to act accordingly. Program rules may have the form:

$$head \leftarrow [selection_condition]\{body\}, \quad (1)$$

where *selection_condition* predicates about some social condition concerning either the cardinality of communities or particular individuals satisfying *body*.

For instance, consider the following rule, belonging to a program representing a given agent **A**: $a \leftarrow [l, h] \{b, not\ c\}$. This rule means that **A** will require **a** only if n agents (other than **A**) exist such that they require or tolerate **b**, do not require or tolerate **c** and it holds that $0 \leq l \leq n \leq h \leq n_{agent} - 1$, where n_{agent} is the total number of agents¹.

This enriched language is referred to as *SOcial Logic Programming* (SOLP). The wedding party example above may be represented by the four SOLP programs shown in Table 1, where the program \mathcal{P}_4 is empty since the corresponding agent has not any desire to express.

The intended models must represent the mental states of each agent inside the community. For instance, the agents' choices w.r.t. the party can be:

$\{\}$, $\{go_wedding_{\mathcal{P}_1}, go_wedding_{\mathcal{P}_2}, drive_{\mathcal{P}_2}\}$, and $\{go_wedding_{\mathcal{P}_1}, go_wedding_{\mathcal{P}_2}, go_wedding_{\mathcal{P}_3}\}$, where the subscript \mathcal{P}_i ($1 \leq i \leq n_{agent}$) references, for each atom in a model, the program (resp. agent) that atom is entailed by. The models respectively mean that either (i) no agent will go to the party, (ii) only **Agent**₁ and **Agent**₂ will go and also **Agent**₂ will drive the car, or (iii) all agents but **Agent**₄ will go to the party.

Indeed, **Agent**₄ anyway does not go. On the one hand, if **Agent**₂ does not go to the party, then **Agent**₃ will do the same. Now, let n' be the number of agents which are going to the party, it is $n' = 0$. **Agent**₁ requires that at least $\nu = \frac{n_{agent}}{2} - 1$ agents (other than himself) go to the party, but since it is $\nu = \frac{4}{2} - 1 = 1$ and $n' < \nu$, then **Agent**₁ does not go to the party (case (i)).

¹ By default, $l = 0$ and $h = n_{agent} - 1$.

$\mathcal{P}_1 (\mathbf{Agent}_1) :$ $\text{go_wedding} \leftarrow [\frac{n_{agent}}{2} - 1,]\{\text{go_wedding}\}$	$\mathcal{P}_2 (\mathbf{Agent}_2) :$ $\text{okay}(\text{go_wedding}) \leftarrow$ $\text{okay}(\text{drive}) \leftarrow \text{go_wedding}$
$\mathcal{P}_3 (\mathbf{Agent}_3) :$ $\text{go_wedding} \leftarrow [\mathbf{Agent}_2]\{\text{go_wedding},$ $\text{not drive}\}$	$\mathcal{P}_4 (\mathbf{Agent}_4) :$ <i>empty program</i>

Table 1. The wedding party example

On the other hand, if \mathbf{Agent}_2 goes to the party, it is possible that he wants either to drive the car or not. If he wants to drive, then \mathbf{Agent}_3 will not join the party. Now, it is $n' = 1 = \nu$, then \mathbf{Agent}_1 will go to the party (case *(ii)*). Otherwise, if \mathbf{Agent}_2 does not want to drive the car, then all conditions required by \mathbf{Agent}_3 are satisfied, thus he will go to the party. Now, it is $n' = 2 > \nu$ and then \mathbf{Agent}_1 will join the party too (case *(iii)*).

The intended models are referred to as *social models*, since they express the results of the interactions among agents.

Our work is strongly related to [5], where the *Joint Fixpoint Semantics* (JFP), that is a semantics providing a way to reach a compromise (in terms of a common agreement) among many agents, is proposed. Therein, each model contains atoms representing items being common to all the agents. Our paper extends such a semantics, providing *feature-selective atom subset community*, i.e. given a set S of SOLP programs representing a community of agents, a program $\mathcal{P} \in S$ and a rule $r \in \mathcal{P}$ of the form $\text{head} \leftarrow [\text{selection_condition}]\{\text{body}\}$, then head will belong to an intended model if all properties enclosed in $\{\text{body}\}$ are entailed by either *(i)* any subset $S' \subseteq (S \setminus \{\mathcal{P}\})$ of programs with a given cardinality (specified by $[\text{selection_condition}]$) or *(ii)* some particular program different from \mathcal{P} .

An example of case *(i)* is shown in Table 1 by the program \mathcal{P}_1 : An intended model M will include the atom $\text{go_wedding}_{\mathcal{P}_1}$ if a set of programs $S' \subseteq \{\mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_4\}$ exists such that $\forall \mathcal{P}_i \in S', \text{go_wedding}_{\mathcal{P}_i} \in M$ and $|S'| \geq \frac{n_{agent}}{2} - 1$. An example of case *(ii)* is represented by the program \mathcal{P}_3 , which requests the atom $\text{go_wedding}_{\mathcal{P}_3}$ to be part of an intended model M if $\text{go_wedding}_{\mathcal{P}_2}$ belongs to M , but the atom $\text{drive}_{\mathcal{P}_2}$ does not.

Importantly, social constraints can be nested. Consider for example the program: $\text{download}(X) \leftarrow [\text{min},]\{\text{shared}(X), [1,]\{\text{not incomplete}(X)\}\}, \text{file}(X)$. This program represents a Peer-to-Peer file-sharing system where a user can share his collection of files with other users on the Internet. In order to get better performances, a file is split into several parts being downloaded separately (possibly

each part from a different user)². Thus, the program describes the behavior of an agent (acting on behalf of a given user) that wants to download any file X being shared by at least a number min of users such that at least one of them owns a complete version of X .

We show also that, given a set of SOLP programs in input, a source-to-source transformation is possible which provides as output a single DLP^A [8] program whose stable models are in one-to-one correspondence with the intended ones. We recall that DLP^A is basically disjunctive logic programming with aggregate functions, supported by the DLV system [9]. The translation to DLP^A give us the ability of exploiting DLV (widely accepted as the state-of-the-art system implementing disjunctive logic programming)³. Observe that since our language includes neither disjunction nor classical negation (even though the extensions to these cases could be considered), both disjunction and classical negation of DLP^A are never enabled by our translation. Moreover, Section 5 shows that our kind of social reasoning is not trivial, since even in the case of positive programs, the semantics of SOLP has a computational complexity which is NP complete.

The paper is organized as follows: in Sections 2 and 3 we respectively define the notion of SOLP programs and define their semantics (*Social Semantics*). In Section 4 we illustrate how a set of SOLP programs, each representing a different agent, is translated into a single DLP^A logic program whose stable models describe the mental states of the whole agent community and then we show that such a translation is correct. In Section 5 we prove that the Social Semantics extends the JFP Semantics [5] and we study the complexity of the problem of searching for a social model. In Section 6 we describe how this novel approach may be used for knowledge representation and finally, we draw our conclusions. For space restrictions, proofs of theorems and lemmata are omitted. They can be found in [4].

2 Social Logic Programs: Basic Definitions

In this section we introduce the notion of SOLP program.

A *term* is either a variable or a constant. An *atom* or *positive literal* is an expression $p(t_1, \dots, t_n)$, where p is a *predicate* of arity n and t_1, \dots, t_n are terms. A *negative literal* is the *negation as failure (NAF) not a* of a given atom a .

Definition 1. A (n -)social selection constraint s , said also (n -)SSC, is an expression of the form $cond(s) \text{ property}(s)$, such that:

- (1) $cond(s)$ is an expression $[\alpha]$ where α is either (i) a pair of integers l, h such that $0 \leq l \leq h \leq n-1$, or (ii) an integer belonging to $\{1, \dots, n\}$ said *program identifier*⁴.

² Among others, KaZaA, EDonkey, WinMX and BitTorrent are the most popular Internet P2P file-sharing systems exploiting such a feature.

³ Of course, our approach may easily be adapted to other systems supporting cardinality constraints, such as Smodels.

⁴ We will show, at the end of the section, that the program identifier uniquely identifies a program (i.e., an agent).

(2) $property(s) = content(s) \cup skel(s)$, where $content(s)$ is a non-empty set of literals and $skel(s)$ is a (possibly empty) set of SSCs.

Concerning item (1) of the above definition, in case (i), $cond(s)$ is said *cardinal selection condition*, while, in case (ii), $cond(s)$ is said *member selection condition*.

n -social selection constraints operate over a collection of n programs (we will formally define later in this section which kind of program are allowed). Thus, with a little abuse of notation, we often denote a member selection condition by $[P_j]$ instead of $[j]$.

Concerning item (2) of Definition 1, if $skel(s) = \emptyset$ then s is said *simple*. For a simple SSC s such that $content(s)$ is singleton, the enclosing braces can be omitted. Finally, given a SSC s , the formula *not s* is said the NAF of s .

In our initial wedding party example, $[\frac{n_{agent}}{2} - 1,]\{go_wedding\}$ and $[Agent_2]\{go_wedding, not\ drive\}$ are two simple SSCs. On the contrary, the SSC occurring in the example regarding a Peer-to-Peer system (see Page 3) is not simple.

As a further example, if $s = [l, h]\{a, b, c, [l_1, h_1]\{d, [l_2, h_2]e\}, [l_3, h_3]f\}$, then s is not simple, $content(s) = \{a, b, c\}$ and $skel(s) = \{[l_1, h_1]\{d, [l_2, h_2]e\}, [l_3, h_3]f\}$.

Now we define a function which returns, for a given SSC s , its nesting depth. Given a SSC s , we define the function $depth$ as follows:

$$\begin{cases} depth(s) = depth(s') + 1, & \text{if } \exists s' \mid s \in skel(s') \\ depth(s) = 0, & \text{otherwise.} \end{cases}$$

Given two SSCs s and s' such that $cond(s) = [l, h]$ and $cond(s') = [l', h']$, i.e. they are cardinal selection conditions, we say that $cond(s') \subseteq cond(s)$ if $h' \leq h$.

A SSC s is *well-formed* if either (i) s is simple, or (ii) s is not simple, $cond(s)$ is a cardinal selection condition and $\forall s' \in skel(s)$ it holds that:

- (a) If $cond(s')$ is a cardinal selection condition, then s' is *well-formed* and $cond(s') \subseteq cond(s)$;
- (b) If $cond(s')$ is a member selection condition, then s' is simple.

From now on, we consider only well-formed SSCs.

Example 1. The SSC $s = [1, 8]\{a, [3, 6]\{b, [Agent_x]\{c, d\}\}\}$ is well-formed, while $s_1 = [4, 7]\{a, [3, 9]b\}$ is not a well-formed SSC, because $[3, 9] \not\subseteq [4, 7]$.

We introduce now the notion of rule. Our definition generalizes the notion of classical logic rule.

Definition 2. A (n -)social rule r is a formula $a \leftarrow b_1 \wedge \dots \wedge b_m \wedge s_1 \wedge \dots \wedge s_k$ ($m \geq 0, k \geq 0$), where a is an atom, each b_i ($1 \leq i \leq m$) is a literal and each s_j ($1 \leq j \leq k$) is either a n -SSC or the NAF of a n -SSC. The atom a is said the *head* of r , while the conjunction $b_1 \wedge \dots \wedge b_m \wedge s_1 \wedge \dots \wedge s_k$ is said the *body* of r . In case a is of the form $okay(p)$, where p is an atom, then r it is said (n -)tolerance (social) rule and p is said the *head* of r . In case $k = 0$, a social non-tolerance rule is said *classical rule*.

Given a rule r , we denote by $head(r)$ (resp. $body(r)$) the head (resp. the body) of r . Moreover, r is said a *fact* in case the body is empty, while r is said an *integrity constraint* if the head is missing.

Definition 3. A SOLP *collection* is a set $\{\mathcal{P}_1, \dots, \mathcal{P}_n\}$ of SOLP programs, where each SOLP *program* is a set of n -social rules. The cardinal i ($1 \leq i \leq n$) is called *program identifier* of the program \mathcal{P}_i .

A SOLP program is *positive* if no NAF symbol *not* occurs in it. For the sake of presentation we only refer, in the following sections, to *ground* (i.e., variable-free) SOLP programs – the extension to the general case is straightforward.

3 Semantics of SOLP programs

In this section we introduce the *Social Semantics*, i.e. the semantics of a collection of SOLP programs. We assume the reader is familiar with the basic concepts of logic programming [1, 12].

We start by introducing the notion of interpretation for a single SOLP program (note that this is the same as for classical programs). An *interpretation* for a ground SOLP program \mathcal{P} is a subset of $Var(\mathcal{P})$, where $Var(\mathcal{P})$ is the set of atoms appearing in \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 .

Before defining the intended models of our semantics, we need some preliminary definitions. Let \mathcal{P} be a SOLP program. We define the *autonomous reduction* of \mathcal{P} , denoted by $A\mathcal{P}$, the program obtained from \mathcal{P} by removing all the SSCs from the rules in \mathcal{P} . Thus, if the program \mathcal{P} represents the social behavior of an agent, then $A\mathcal{P}$ represents the behavior of the same agent in case he decides to operate independently of the other agents.

Given a SOLP program \mathcal{P} and an interpretation $I \subseteq Var(A\mathcal{P})$, let $CL(A\mathcal{P})$ (resp. $TR(A\mathcal{P})$) be the set of classical (resp. tolerance) rules in $A\mathcal{P}$. The *autonomous immediate consequence operator* $AT_{\mathcal{P}}$ is the function from $2^{Var(A\mathcal{P})}$ to $2^{Var(A\mathcal{P})}$ defined as follows:

$$AT_{\mathcal{P}}(I) = \{head(r) \mid \forall r \in CL(A\mathcal{P}), body(r) \text{ is true w.r.t. } I\} \cup \{head(r) \mid \forall r \in TR(A\mathcal{P}), body(r) \wedge head(r) \text{ is true w.r.t. } I\}.$$

Definition 4. An interpretation I for a SOLP program \mathcal{P} is an *autonomous fixpoint* of \mathcal{P} if I is a fixpoint of the associated transformation $AT_{\mathcal{P}}$, i.e. if $AT_{\mathcal{P}}(I) = I$. The set of all autonomous fixpoints of \mathcal{P} is denoted by $AFP(\mathcal{P})$.

Thus, the autonomous fixpoints of a given SOLP program \mathcal{P} represent the mental states of the corresponding agent, whenever every social constraint in \mathcal{P} is discarded.

Definition 5. Let \mathcal{P} be a SOLP program and L be a set of literals. The *labeled version* of L w.r.t. \mathcal{P} is the set $L_{\mathcal{P}} = \{a_{\mathcal{P}} \mid a \in L\}$.

Let $C = \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$ be a SOLP collection. A *social interpretation* for C is a set $\bar{I} = I_{\mathcal{P}_1}^1 \cup \dots \cup I_{\mathcal{P}_n}^n$, where I^j is an interpretation for \mathcal{P}_j ($1 \leq j \leq n$).

Example 2. If $C = \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}$, $I^1 = \{a, b, c\}$, $I^2 = \{a, d, e\}$ and $I^3 = \{b, c, d\}$, where I^j is an interpretation for \mathcal{P}_j ($1 \leq j \leq 3$), then $\bar{I} = \{a_{\mathcal{P}_1}, b_{\mathcal{P}_1}, c_{\mathcal{P}_1}, a_{\mathcal{P}_2}, d_{\mathcal{P}_2}, e_{\mathcal{P}_2}, b_{\mathcal{P}_3}, c_{\mathcal{P}_3}, d_{\mathcal{P}_3}\}$ is a social interpretation for C .

Let $C = \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$ be a SOLP collection and $\mathcal{P} \in C$. Given a social interpretation \bar{I} for C , a positive literal $a \in \text{Var}(\mathcal{P})$ (resp. a negative literal $\text{not } a$) is *true* w.r.t. \bar{I} if $a_{\mathcal{P}} \in \bar{I}$ (resp. $a_{\mathcal{P}} \notin \bar{I}$); otherwise it is *false*.

Before giving the definition of truth for a SSC, we introduce a way to reference any SSC s (and also every SSC nested in s) occurring in a given rule r of a SOLP program \mathcal{P} .

Given a SOLP program \mathcal{P} , a social rule $r \in \mathcal{P}$ and an integer $n \geq 0$, we define the set $MSSC^{(\mathcal{P}, r, n)} = \{s \mid s \text{ is a SSC occurring in } r \in \mathcal{P} \wedge \text{depth}(s) = n\}$. Observe that $MSSC^{(\mathcal{P}, r, 0)}$ denotes the set of SSCs as they appear in the rule r of the SOLP program \mathcal{P} .

Example 3. Let $a \leftarrow [1, 8]\{a, [3, 6]\{b, [\mathbf{Agent}_x]\{c, d\}\}, [2, 3]\{e, f\}$ be a rule r in a SOLP program \mathcal{P} . Then:

$$\begin{aligned} MSSC^{(\mathcal{P}, r, 0)} &= \{ [1, 8]\{a, [3, 6]\{b, [\mathbf{Agent}_x]\{c, d\}\}, [2, 3]\{e, f\} \}, \\ MSSC^{(\mathcal{P}, r, 1)} &= \{ [3, 6]\{b, [\mathbf{Agent}_x]\{c, d\}\} \}, \\ MSSC^{(\mathcal{P}, r, 2)} &= \{ [\mathbf{Agent}_x]\{c, d\} \}, \\ MSSC^{(\mathcal{P}, r, 3)} &= \emptyset. \end{aligned}$$

Given a SOLP program \mathcal{P} , we define the set $MSSC^{\mathcal{P}} = \bigcup_{r \in \mathcal{P}} MSSC^{(\mathcal{P}, r, 0)}$.

Thus $MSSC^{\mathcal{P}}$ is the set of all the SSCs (with depth 0) occurring in \mathcal{P} .

Now we provide the definition of truth of a SSC w.r.t. a given social interpretation and, subsequently, the definition of truth of a social rule.

Definition 6. Let $C = \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$ be a SOLP collection, $C' \subseteq C$ and $\mathcal{P}_j \in C'$. Given a social interpretation \bar{I} for C' and a n -SSC $s \in MSSC^{\mathcal{P}_j}$, we say that s is *true* in C' w.r.t. \bar{I} if it holds that either:

- (1) $\text{cond}(s) = [k] \wedge$
 $\exists \mathcal{P}_k \in C' \mid \forall a_{\mathcal{P}_k} \in (\text{content}(s))_{\mathcal{P}_k} \quad a \text{ is true w.r.t. } \bar{I}, \text{ or}$
- (2) $\text{cond}(s) = [l, h] \wedge$
 $\exists D \subseteq C' \setminus \{\mathcal{P}_j\} \mid l \leq |D| \leq h \wedge$
 $\forall a_{\mathcal{P}} \in \bigcup_{\mathcal{P} \in D} (\text{content}(s))_{\mathcal{P}} \quad a \text{ is true w.r.t. } \bar{I} \wedge$
 $\forall s' \in \text{skel}(s) \exists D' \subseteq D \mid s' \text{ is true in } D' \text{ w.r.t. } \bar{I},$

where l, h and k are integers (observe that k is a program identifier). If $C' = C$, then we simply say that s is *true* w.r.t. \bar{I} . A n -SSC not true (in C') w.r.t. \bar{I} is said *false* (in C') w.r.t. \bar{I} .

Finally, the NAF of a n -SSC s , *not* s , is said *true* (resp. *false*) (in C') w.r.t. \bar{I} if s is false (resp. true) (in C') w.r.t. \bar{I} .

Thus, given a SSC s included in \mathcal{P}_j , s is true w.r.t. a social interpretation \bar{I} if a single SOLP program corresponding to a program identifier k (resp. a set of

SOLP programs) exists (resp. not including \mathcal{P}_j) such that all the elements in $property(s)$ are true w.r.t. \bar{I} . Observe that the truth of $property(s)$ w.r.t. \bar{I} is possibly defined recursively, since s may contain nested SSCs.

Once the notion of truth of SSCs has been defined, we are able to define the notion of satisfaction of a social rule w.r.t. a social interpretation.

Let $C = \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$ be a SOLP collection and $\mathcal{P} \in C$. Given a social interpretation \bar{I} for C , a social rule in \mathcal{P} is *satisfied* (or is *true*) w.r.t. \bar{I} if its head is true w.r.t. \bar{I} or its body is false w.r.t. \bar{I} .

Given a SOLP collection $\{\mathcal{P}_1, \dots, \mathcal{P}_n\}$, we define the set of *candidate social interpretations* for $\mathcal{P}_1, \dots, \mathcal{P}_n$ as

$$\mathcal{U}(\mathcal{P}_1, \dots, \mathcal{P}_n) = \{F_{\mathcal{P}_1}^1 \cup \dots \cup F_{\mathcal{P}_n}^n \mid F^i \in AFP(\mathcal{P}_i), 1 \leq i \leq n\}.$$

where, recall, $AFP(\mathcal{P}_i)$ is the set of autonomous fixpoints of the SOLP program \mathcal{P}_i , introduced in Definition 4 and by $F_{\mathcal{P}}^i$ ($1 \leq i \leq n$) we denote the labeled version of F^i w.r.t. \mathcal{P} (see Definition 5). $\mathcal{U}(\mathcal{P}_1, \dots, \mathcal{P}_n)$ represents all the configurations obtained by combining the autonomous (i.e. without considering the social constraints) mental states of the agents corresponding to the programs $\mathcal{P}_1, \dots, \mathcal{P}_n$. Each candidate social interpretation is a candidate intended model.

The intended models are then obtained by enabling the social constraints.

Now, we are ready to give the definition of intended model w.r.t. the Social Semantics.

Definition 7. Given a SOLP collection $C = \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$, a candidate social interpretation \bar{I} for C is a *social model* of C if $\forall r \in \bigcup_{1 \leq i \leq n} \mathcal{P}_i$, r is true w.r.t. \bar{I} .

Definition 8. Given a SOLP collection $\{\mathcal{P}_1, \dots, \mathcal{P}_n\}$, the *Social Semantics* (often referred to as *S-Semantics*) of $\mathcal{P}_1, \dots, \mathcal{P}_n$ is the set

$$\mathcal{SOS}(\mathcal{P}_1, \dots, \mathcal{P}_n) = \{\bar{M} \mid \bar{M} \in \mathcal{U}(\mathcal{P}_1, \dots, \mathcal{P}_n) \wedge \bar{M} \text{ is a social model of } \mathcal{P}_1, \dots, \mathcal{P}_n\},$$

Thus $\mathcal{SOS}(\mathcal{P}_1, \dots, \mathcal{P}_n)$ is the set of all social models of $\mathcal{P}_1, \dots, \mathcal{P}_n$.

Given a SOLP collection $C = \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$ and a SOLP program $\mathcal{P} \in C$, we define the *S-Semantics* of \mathcal{P} as

$$\mathcal{S}(\mathcal{P}) = \{F \mid F \in AFP(\mathcal{P}) \wedge \exists \bar{M} \in \mathcal{SOS}(\mathcal{P}_1, \dots, \mathcal{P}_n) \mid F_{\mathcal{P}} \subseteq \bar{M}\},$$

Hence $\mathcal{S}(\mathcal{P})$ represents the autonomous mental states of an agent, corresponding to a SOLP program \mathcal{P} , which are also included in some social model of $\mathcal{P}_1, \dots, \mathcal{P}_n$, and then fulfill all social requirements.

4 Translation

In this section we give the translation from SOLP under the Social Semantics to DLP^A [8] under Stable Model Semantics. We assume that the reader is familiar with the Stable Model Semantics [10]. Given a classical logic program \mathcal{P} , we denote by $SM(\mathcal{P})$ the set of all the stable models of \mathcal{P} .

Given a SOLP program \mathcal{P} , we define the set $USSC^{\mathcal{P}} = \bigcup_{r \in \mathcal{P}} \bigcup_{n \geq 0} MSSC^{\langle \mathcal{P}, r, n \rangle}$.

Thus, $USSC^{\mathcal{P}}$ includes all the SSCs (at any nesting depth) in \mathcal{P} .

Given a SOLP program \mathcal{P} , we define the functions ρ and g , each establishing a one-to-one correspondence between each element in $USSC^{\mathcal{P}}$ and a set of atoms L such that both (i) $L \cap Var(\mathcal{P}) = \emptyset$ and (ii) $\forall s, t \in USSC^{\mathcal{P}}, \rho(s) \neq g(t)$. Thus, given a SSC s included in a SOLP program \mathcal{P} , $\rho(s)$ is a unique positive literal identifying s and we denote by $(\rho(s))_{\mathcal{P}}$ the labeled version of $\rho(s)$ w.r.t. \mathcal{P} . Similar considerations hold for $g(s)$. Moreover, with a little abuse of notation we write $(g(s))_{\mathcal{P}}(x)$ denoting the labeled version of the predicate $(g(s))(x)$ w.r.t. \mathcal{P} .

Now we introduce the translation of a single SSC and then we extend such a translation to all SSCs included in a social rule, a SOLP program and a SOLP collection, respectively.

Definition 9. Given a SOLP collection $\{\mathcal{P}_1, \dots, \mathcal{P}_j, \dots, \mathcal{P}_n\}$ and $s \in USSC^{\mathcal{P}_j}$, we define the *translation of s* as the DLP^A program $\Psi^{\mathcal{P}_j}(s) = GUESS^{\mathcal{P}_j}(s) \cup CHECK^{\mathcal{P}_j}(s)$, where $GUESS^{\mathcal{P}_j}(s) =$

$$= \begin{cases} \{(g(s))_{\mathcal{P}_j}(k) \leftarrow \bigwedge_{b \in content(s)} b_{\mathcal{P}_k}\}, & \text{if } cond(s) = [k], \\ \{(g(s))_{\mathcal{P}_j}(i) \leftarrow \bigwedge_{b \in content(s)} b_{\mathcal{P}_i} \wedge \\ \bigwedge_{s' \in skel(s)} (g(s'))_{\mathcal{P}_j}(i) \mid \\ 1 \leq i \neq j \leq n\} \cup \\ \{GUESS^{\mathcal{P}_j}(s') \mid s' \in skel(s)\}, & \text{if } cond(s) = [l, h], \end{cases}$$

and $CHECK^{\mathcal{P}_j}(s) =$

$$= \begin{cases} \{(\rho(s))_{\mathcal{P}_j} \leftarrow (g(s))_{\mathcal{P}_j}(k)\}, & \text{if } cond(s) = [k], \\ \{(\rho(s))_{\mathcal{P}_j} \leftarrow l \leq \#count\{K : (g(s))_{\mathcal{P}_j}(K), K \neq j\} \leq h\} \cup \\ \{CHECK^{\mathcal{P}_j}(s') \mid s' \in skel(s)\}, & \text{if } cond(s) = [l, h], \end{cases}$$

where $\#count$ is an aggregate function which returns the cardinality of a set of literals satisfying some conditions [8]. Observe that the above translation produces a safe aggregate-stratified DLP^A program and thus the computational complexity remains the same as for standard DLP [8, 9].

Given a SOLP program \mathcal{P}_j and a social rule $r \in \mathcal{P}_j$, we define the *SSC translation of r* as the DLP^A program $T^{\mathcal{P}_j}(r) = \bigcup_{s \in MSSC^{\langle \mathcal{P}, r, 0 \rangle}} \Psi^{\mathcal{P}_j}(s)$. Observe that, for any classical rule $r \in \mathcal{P}_j$, it holds that $T^{\mathcal{P}_j}(r) = \emptyset$.

Given a SOLP program \mathcal{P}_j , the *SSC translation of \mathcal{P}_j* is the DLP^A program $W^{\mathcal{P}_j} = \bigcup_{r \in \mathcal{P}_j} T^{\mathcal{P}_j}(r)$.

Definition 10. Given a SOLP collection $\{\mathcal{P}_1, \dots, \mathcal{P}_n\}$, we define the *SSC translation of the collection* as the DLP^A program $C(\mathcal{P}_1, \dots, \mathcal{P}_n) = \bigcup_{1 \leq i \leq n} W^{\mathcal{P}_i}$.

Thus $C(\mathcal{P}_1, \dots, \mathcal{P}_n)$ is a DLP^A program representing the translation of all the SSCs included in $\mathcal{P}_1, \dots, \mathcal{P}_n$.

We have defined above the translation for the SSCs included in a SOLP program. Now we give the translation for the whole SOLP program, but first we need a preliminary processing of all tolerance rules. The latter is done by means of the transformation defined as follows:

Given a SOLP program \mathcal{P} , $\hat{\mathcal{P}} = \mathcal{P} \setminus TR(\mathcal{P}) \cup \{head(r) \leftarrow head(r) \wedge body(r) \mid r \in TR(\mathcal{P})\}$. Thus $\hat{\mathcal{P}}$ is obtained from \mathcal{P} by replacing each tolerance rule $okay(p) \leftarrow body$ with the rule $p \leftarrow p, body$.

Definition 11. Let \mathcal{P} be a SOLP program. We define the program $\Gamma'(\hat{\mathcal{P}})$ over the set of atoms $Var(\Gamma'(\hat{\mathcal{P}})) = \{a_{\mathcal{P}} \mid a \in Var(A\hat{\mathcal{P}})\} \cup \{a'_{\mathcal{P}} \mid a \in Var(A\hat{\mathcal{P}})\} \cup \{sa_{\mathcal{P}} \mid a \in Var(A\hat{\mathcal{P}})\} \cup \{fail_{\mathcal{P}}\}$ as $\Gamma'(\hat{\mathcal{P}}) = S'_1(\hat{\mathcal{P}}) \cup S'_2(\hat{\mathcal{P}}) \cup S'_3(\hat{\mathcal{P}})$, where $S'_1(\hat{\mathcal{P}})$, $S'_2(\hat{\mathcal{P}})$ and $S'_3(\hat{\mathcal{P}})$ are defined as follows:

$$\begin{aligned} S'_1(\hat{\mathcal{P}}) &= \{a_{\mathcal{P}} \leftarrow not\ a'_{\mathcal{P}} \mid a \in Var(A\hat{\mathcal{P}})\} \cup \{a'_{\mathcal{P}} \leftarrow not\ a_{\mathcal{P}} \mid a \in Var(A\hat{\mathcal{P}})\}, \\ S'_2(\hat{\mathcal{P}}) &= \{sa_{\mathcal{P}} \leftarrow b^1_{\mathcal{P}}, \dots, b^n_{\mathcal{P}}, (\rho(s_1))_{\mathcal{P}}, \dots, (\rho(s_m))_{\mathcal{P}} \mid a \leftarrow b_1, \dots, b_n, s_1, \dots, s_m \in \mathcal{P}\}, \\ S'_3(\hat{\mathcal{P}}) &= \{fail_{\mathcal{P}} \leftarrow not\ fail_{\mathcal{P}}, sa_{\mathcal{P}}, not\ a_{\mathcal{P}} \mid a \in Var(A\hat{\mathcal{P}})\} \cup \\ &\quad \{fail_{\mathcal{P}} \leftarrow not\ fail_{\mathcal{P}}, a_{\mathcal{P}}, not\ sa_{\mathcal{P}} \mid a \in Var(A\hat{\mathcal{P}})\}. \end{aligned}$$

Definition 12. Given a SOLP collection $\{\mathcal{P}_1, \dots, \mathcal{P}_n\}$, we define the set $P'_u = \bigcup_{1 \leq i \leq n} \Gamma'(\hat{\mathcal{P}}_i)$.

Thus P'_u is a classical logic program representing the translation of the SOLP collection. This program is used in conjunction with the DLP^A program $C(\mathcal{P}_1, \dots, \mathcal{P}_n)$ in order to enable the social constraints.

In the next theorem we state that a one-to-one correspondence exists between the social models in $SOS(\mathcal{P}_1, \dots, \mathcal{P}_n)$ and the stable models of the DLP^A program $P'_u \cup \bar{C}$. First, we need the following definition and results:

Let \mathcal{P} be a SOLP program and $M \subseteq Var(\mathcal{P})$. We denote by $[M]_{\mathcal{P}}$ the set $\{a_{\mathcal{P}} \mid a \in M\} \cup \{a'_{\mathcal{P}} \mid a \in Var(\mathcal{P}) \setminus M\} \cup \{sa_{\mathcal{P}} \mid a \in M\}$.

Lemma 1. Given a SOLP collection $SP = \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$, a social interpretation \bar{I} for SP , a SOLP program $\mathcal{P}_j \in SP$ and a SSC $s \in MSSC^{\mathcal{P}_j}$, assume $\bar{C} = C(\mathcal{P}_1, \dots, \mathcal{P}_n)$ and $Q = \{a \leftarrow \mid a \in \bar{I}\}$. Then:

$$s \text{ is true w.r.t. } \bar{I} \text{ iff } \exists M \in SM(\bar{C} \cup Q) \mid (\rho(s))_{\mathcal{P}_j} \in M.$$

Definition 13. Given a SOLP collection $SP = \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$, a social interpretation \bar{I} for SP , a SOLP program $\mathcal{P}_j \in SP$ and a SSC $s \in MSSC^{\mathcal{P}_j}$, assume $\bar{C} = C(\mathcal{P}_1, \dots, \mathcal{P}_n)$ and $Q = \{a \leftarrow \mid a \in \bar{I}\}$. We define the set

$$SAT_{\bar{I}}^{\mathcal{P}_j}(s) = \{h \mid h = head(r), r \in \Psi^{\mathcal{P}_j}(s) \wedge \exists M \in SM(\bar{C} \cup Q) \mid h \in M\}.$$

Thus, by virtue of Lemma 1, if s is true w.r.t. \bar{I} , then $SAT_{\bar{I}}^{\mathcal{P}_j}(s)$ includes the literal $(\rho(s))_{\mathcal{P}_j}$ and those heads of rules in $\Psi^{\mathcal{P}_j}(s)$ corresponding (by means of the functions ρ and g) to both s and the SSCs which are nested in s .

Theorem 1. Given a SOLP collection $\{\mathcal{P}_1, \dots, \mathcal{P}_n\}$, and $\bar{C} = C(\mathcal{P}_1, \dots, \mathcal{P}_n)$. Then $A = B$, where:

$$\begin{aligned}
A &= SM(P'_u \cup \bar{C}) && \text{and} \\
B &= \{ \bar{F} \cup \bar{G} \cup \bar{H} \mid \\
&\quad \bar{F} = \bigcup_{1 \leq i \leq n} F_{\mathcal{P}_i}^i \wedge F^i \in AFP(\mathcal{P}_i) \wedge \bar{F} \in SOS(\mathcal{P}_1, \dots, \mathcal{P}_n) \\
&\quad \bar{G} = \bigcup_{1 \leq i \leq n} G_{\mathcal{P}_i}^i \wedge G_{\mathcal{P}_i}^i = [F^i]_{\mathcal{P}_i} \setminus F_{\mathcal{P}_i}^i \\
&\quad \bar{H} = \bigcup_{1 \leq i \leq n} H_{\mathcal{P}_i}^i \wedge H_{\mathcal{P}_i}^i = \bigcup_{s \in MSSC^{\mathcal{P}_i}} SAT_{\bar{F}}^{\mathcal{P}_i}(s) \}.
\end{aligned}$$

Thus each stable model $x \in A$ may be partitioned in three sets: \bar{F} (the social model of the SOLP collection, which corresponds to x), \bar{G} and \bar{H} (both including overhead literals needed by the translation).

5 Social Models, Joint Fixpoints and Complexity

In this section we show that the Social Semantics extends the JFP semantics [5]. Basically, COLP programs are logic programs containing also *tolerance* rules, that are rules of the form $okay(p) \leftarrow body(r)$. The semantics of a collection of COLP programs is defined over classical programs obtained by the COLP programs by translating each rule of the form $okay(p) \leftarrow body(r)$ into the rule $p \leftarrow p, body(r)$. The semantics of a collection $\mathcal{P}_1, \dots, \mathcal{P}_n$ of COLP programs is defined in [5] in terms of joint (i.e., common) fixpoints (of the *immediate consequence operator*) of the logic programs obtained from $\mathcal{P}_1, \dots, \mathcal{P}_n$ by transforming *tolerance* rules occurring in them (as shown above). Recall that 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 $I \subseteq Var(\mathcal{P})$, $T_{\mathcal{P}}(I)$ is the set of all heads of rules in \mathcal{P} whose bodies are true w.r.t. I .

First, we define a translation from COLP programs [5] to SOLP programs:

Definition 14. Given a COLP program \mathcal{P} and an integer $n \geq 1$, the *SOLP translation* of \mathcal{P} is a SOLP program $\sigma^n(\mathcal{P}) = \{\sigma_{rule}(r) \mid r \in \mathcal{P}\}$, where

$$\sigma_{rule}(r) = \begin{cases} head(r) \leftarrow [n-1, n-1]head(r), body(r) & \text{if } r \text{ is a classical rule,} \\ okay(p) \leftarrow [n-1, n-1]p, body(r) & \text{if } head(r) = okay(p). \end{cases}$$

The next theorem states that the JFP semantics is a special case of the Social Semantics. $JFP(\mathcal{P}_1, \dots, \mathcal{P}_n)$ denotes the set of the joint fixpoints of $\mathcal{P}_1, \dots, \mathcal{P}_n$.

Theorem 2. Given $n \geq 1$, let $\mathcal{P}_1, \dots, \mathcal{P}_n$ be COLP programs and Q_1, \dots, Q_n be SOLP programs such that $Q_i = \sigma^n(\mathcal{P}_i)$. Then:

$$SOS(Q_1, \dots, Q_n) = \left\{ \bigcup_{1 \leq i \leq n} F_{Q_i} \mid F \in JFP(\mathcal{P}_1, \dots, \mathcal{P}_n) \right\}.$$

Now we introduce a relevant decision problem w.r.t. the Social Semantics and discuss its complexity. Observe that the analysis is done in case of positive programs. Indeed, the case of non positive programs is straightforward: Since it is NP complete to determine whether a *single* non-positive program has a fixpoint, it is easy to see that the same holds for non-positive SOLP programs and autonomous fixpoints. Thus, checking whether a SOLP collection containing at least one non-positive SOLP program has a social model is trivially NP hard. Moreover, since this problem is easily seen to be in NP, it is NP complete.

PROBLEM SOS_n (social model existence):**Instance:** A SOLP collection $\mathcal{P}_1, \dots, \mathcal{P}_n$ **Question:** Is $\mathcal{SOS}(\mathcal{P}_1, \dots, \mathcal{P}_n) \neq \emptyset$, i.e., does the SOLP collection $\mathcal{P}_1, \dots, \mathcal{P}_n$ have any social model?**Theorem 3.** The problem SOS_n is NP complete.**6 Knowledge Representation with SOLP programs**

In this section, we provide two real-life examples showing the capability of our language of representing common knowledge.

Example 4. Seating. We must arrange a seating for a number n_{agent} of agents, with m tables and a maximum of c chairs per table. Agents who like (resp. dislike) each other should (resp. should not) sit at the same table. Moreover, an agent can express some requirements w.r.t. the number and the identity of other agents sitting at the same table. Assume that the i -th agent is represented by a predicate $agent(i)$ ($1 \leq i \leq n_{agent}$) and his knowledge base is included in a single SOLP program. The predicate $like(i)$ (resp. $dislike(i)$) means that $Agent_i$ is desired (resp. not tolerated) at the same table, $table(T)$ represents a table ($1 \leq T \leq m$) and $at(T)$ expresses the desire to sit at table T . For instance, the program \mathcal{P}_1 (which is associated to $Agent_1$) could be written as follows:

$$\begin{array}{ll}
r_1 : & agent(1) \leftarrow \\
r_2 : & \leftarrow at(T1), at(T2), T1 <> T2 \\
r_3 : & at(T) \leftarrow [, c - 1]\{at(T), agent(P)\}, like(P), table(T) \\
r_4 : & \leftarrow at(T), [1,]\{at(T), agent(P)\}, dislike(P) \\
r_5 : & \leftarrow like(P), dislike(P) \\
r_6 : & like(2) \leftarrow \\
r_7 : & dislike(3) \leftarrow \\
r_8 : & okay(like(4)) \leftarrow \\
r_9 : & \leftarrow at(T), [3,]\{at(T)\}
\end{array}$$

where rules from r_1 to r_5 are common to all the programs (of course, the argument of $agent()$ in r_1 is suited to the enclosing program) and rules from r_6 to r_9 express agent's own requirements. In particular, while the rule r_2 states that any agent cannot be seated at more than one table, the rules r_3 and r_4 mean that an agent wants to share the table with no more than $c - 1$ agents he likes and with no agent he dislikes, respectively. The rule r_5 provides consistency for $like$ and $dislike$. The rule r_8 is used to declare that $Agent_1$ tolerates $Agent_4$, i.e. $Agent_4$ possibly shares a table with $Agent_1$, and finally the rule r_9 means that $Agent_1$ does not want to share a table with 3 agents or more.

Observe that while the rule r_3 generates possible seating arrangements, the rules r_2 , r_4 and r_9 discard those which are not allowed.

Example 5. Room arrangement. Consider a house having m rooms. We have to distribute some objects (i.e. furniture and appliances) over the rooms without exceeding the maximum number of objects, say c , allowed per each room. Further constraints may be set w.r.t. the color and/or the type of objects in the same room, etc. Each object is represented by a single SOLP program encoding both the properties of the object and the constraints we want to meet. As an example consider the following program, representing an object *cupboard*:

$$\begin{aligned}
 r_1 &: \text{name}(\text{cupboard}) \leftarrow \\
 r_2 &: \text{type}(\text{furniture}) \leftarrow \\
 r_3 &: \text{color}(\text{yellow}) \leftarrow \\
 r_4 &: \leftarrow \text{at}(R1), \text{at}(R2), R1 \langle \rangle R2 \\
 r_5 &: \text{at}(R) \leftarrow [, 2] \{ \text{at}(R), \text{type}(\text{appliance}), \text{not color}(\text{yellow}), \\
 & \quad [1, 1] \{ \text{name}(\text{fridge}) \} \}, \text{room}(R) \\
 r_6 &: \text{at}(R) \leftarrow [, c - 3] \{ \text{at}(R), \text{type}(\text{furniture}) \}, \text{room}(R) \\
 r_7 &: \leftarrow \text{at}(R), [1,] \{ \text{at}(R), \text{color}(\text{green}) \}
 \end{aligned}$$

where the properties of the object are encoded as predicates representing the *name* (fridge, cupboard, table, etc.), the *type* (furniture or appliance), the *color* and so on. In particular, the rule r_4 states that an object may not be in more than one room, while by means of the rule r_5 , we allow no more than 2 appliances to share the room with the cupboard, provided that they are not yellow and also that one of them is a fridge. The rule r_6 , means that we want the cupboard to be in the same room with any other pieces of furniture, but no more than $c - 3$, in order not to exceed the maximum number of objects. Finally, the rule r_7 states that the cupboard cannot share the room with any green object.

7 Conclusions

In this work we have proposed a new language, *Social Logic Programming* (SOLP), which enables social behavior among a community of agents represented by logic programs, extending COLP [5]. Thus, the intended models of the *Social Semantics* represent those mental states satisfying social requirements imposed by the agents. Moreover, we have given a translation from SOLP to logic programming with aggregates and discussed the computational complexity of SOLP, which has been proved to be NP complete.

References

1. C. Baral and M. Gelfond. Logic Programming and Knowledge Representation. *Journal of Logic Programming*, 19/20:73–148, 1994.
2. M. Bratman. *Intention, Plans, and Practical Reason*. Harvard Univ. Press, 1987.
3. M. Bratman, D. J. Israel, and M. E. Pollack. Plans and Resource-Bounded Practical Reason. AI center technical note SRI-AI 425R, SRI International, 1988.
4. F Buccafurri and G. Caminiti. A Social-Oriented Semantics for Multi-Agent Systems. Technical Report - TR Lab. Ing. Inf. 05/01, DIMET - University of Reggio Calabria, 2005. Available from the authors.

5. F. Buccafurri and G. Gottlob. *Multiagent Compromises, Joint Fixpoints, and Stable Models*, volume 2407 of *LNCS and LNAI*. Springer, 2002.
6. P. R. Cohen and H. Levesque. Rational interaction as the basis for communication. In *Intentions in Communication*. MIT Press, 1990.
7. R. S. Cost, T. Finin, and Y. Labrou. Coordinating Agents Using ACL Conversations. In *Coordination of Internet Agents: Models, Technologies, and Applications*, pages 183–196, 2001.
8. T. Dell’Armi, W. Faber, G. Ielpa, N. Leone, and G. Pfeifer. Aggregate Functions in Disjunctive Logic Programming: Semantics, Complexity, and Implementation in DLV. In *IJCAI-03, Proc. of the 18th Int. Joint Conf. on Artificial Intelligence*, pages 847–852, 2003.
9. T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative problem-solving in DLV. In *Logic-Based Artificial Intelligence*, pages 79–103. Kluwer, 2000.
10. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *5th Conf. on Logic Programming*, pages 1070–1080. MIT Press, 1988.
11. V. Mascardi, M. Martelli, and L. Sterling. Logic-based specification languages for intelligent software agents. *TPLP*, 4(4):429–494, 2004.
12. J. Minker and D. Seipel. *Disjunctive Logic Programming: A Survey and Assessment*, volume 2407 of *LNCS and LNAI*. Springer, 2002.
13. R. S. Patil, R. E. Fikes, P. F. Patel-Schneider, D. McKay, T. Finin, T. Gruber, and R. Neches. The DARPA knowledge sharing effort: progress report. In *Principles of KR and Reasoning: Proc. of the 3rd Int. Conf.* Kaufmann, 1992.
14. W. van der Hoek and W. Wooldrige. Towards a logic of rational agency. *Logic Journal of the IGPL*, 11(2):135–159, 2003.
15. M. Wooldridge. *Reasoning about Rational Agents*. Intelligent Robots and Autonomous Agents. MIT Press, Cambridge, Massachusetts, 2000.
16. M. Wooldridge and N. R. Jennings. Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review*, 2(10):115–152, 1995.