TLP **8** (5 & 6): 643–690, 2008. © 2008 Cambridge University Press
doi:10.1017/S1471068408003463 First published online 11 July 2008 Printed in the United Kingdom

Logic programming with social features[1]

FRANCESCO BUCCAFURRI and GIANLUCA CAMINITI
DIMET—Università "Mediterranea" degli Studi di Reggio Calabria
via Graziella, loc. Feo di Vito, 89122 Reggio Calabria, Italia
(*e-mail:* `bucca@unirc.it`, `gianluca.caminiti@unirc.it`)

submitted 16 January 2007; revised 13 November 2007, 18 April 2008; accepted 21 May 2008

Abstract

In everyday life it happens that a person has to reason out what other people think and how they behave, in order to achieve his goals. In other words, an individual may be required to adapt his behavior by reasoning about the others' mental state. In this paper we focus on a knowledge-representation language derived from logic programming which both supports the representation of mental states of individual communities and provides each with the capability of reasoning about others' mental states and acting accordingly. The proposed semantics is shown to be translatable into stable model semantics of logic programs with aggregates.

KEYWORDS: Logic programming, stable model semantics, knowledge representation

1 Introduction

In everyday life it happens that a person has to reason out what other people think and how they behave, in order to achieve his goals. In other words, an individual may be required to adapt his behavior by reasoning about the others' mental state. This typically happens in the context of cooperation and negotiation: for instance, an individual can propose his own goals if he knows that they would be acceptable to the others. Otherwise he can decide not to make them public. As a consequence, one can increase the success chances of his actions, by having information about the other individuals' knowledge.

In this paper, we focus on a knowledge-representation language derived from logic programming which both supports the representation of mental states of individual communities and provides each with the capability of reasoning about others' mental states and acting accordingly. The proposed semantics is shown to be translatable into stable model semantics of logic programs with aggregates.

We give the flavor of the proposal by two introductory examples, wherein we describe the features of our approach in an informal, yet deep fashion. Even though

[1] An abridged version of this paper appears in (Buccafurri and Caminiti 2005).

in these examples, as well as elsewhere in the paper, we use the term *agent* to denote the individual reasoning, we remark that our focus is basically concerning to the knowledge-representation aspects, with no intention to investigate how this *reasoning layer* could be exploited in the intelligent-agent contexts. However, in Section 8, we relate our work with some conceptual aspects belonging to this research field.

Now consider the first example.

Example 1
There are four agents which have been invited to the same wedding party. Following are the desires of the agents:

Agent$_1$ will go to the party only if at least the half of the total number of agents (not including himself) goes there.

Agent$_2$ possibly does not go to the party, but he tolerates such an option. In case he goes, then he possibly drives the car.

Agent$_3$ would like to join the party together with **Agent**$_2$, but he does not trust on driving skill of **Agent**$_2$. As a consequence, he decides to go to the party only if **Agent**$_2$ goes there and does not want to drive the car.

Agent$_4$ does not go to the party.

Now, assume that 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 not require, yet tolerate some options.

The standard approach to representing communities by means of logic-based agents (Subrahmanian *et al.*, 2000; Costantini and Tocchio, 2002; Satoh and Yamamoto, 2002; Alberti *et al.*, 2004; De Vos *et al.*, 2005) is founded on suitable extensions of logic programming with negation as failure (*not*) where each agent is represented by a single program whose intended models (under a suitable semantics) are the agent's desires/requests. Although, we take this as a starting point, it is still not suitable to model the above example because of two following issues:

(1) There is no natural representation for tolerated options, i.e. options which are not requested, but possibly accepted (see **Agent**$_2$).
(2) A machinery is missing which enables one agent to reason about the behavior of other agents (see **Agent**$_1$ and **Agent**$_3$).

In order to solve the first issue (1) we use an extension of standard logic programming exploiting the special predicate *okay*(), previously introduced in (Buccafurri and Gottlob, 2002). Therein a model-theoretic semantics aimed to represent a common agreement in a community of agents was given. However, representing the requests/acceptances of single agents in a community is not enough. Concerning (2), a social language should also provide a machinery to model possible interference among agents' reasoning (in fact it is just such an interference that distinguishes the social reasoning from the individual one). To this aim, we introduce a new construct providing one agent with the ability to reason about other agents' mental state and then to act accordingly.

Table 1. *The wedding party (Example 1)*

$\mathscr{P}_1$ (**Agent**$_1$):	go_wedding ←	$[\frac{n}{2}-1,]\{go\_\text{wedding}\}$
$\mathscr{P}_2$ (**Agent**$_2$):	*okay*(go_wedding) ←	
	okay(drive) ←	go_wedding
$\mathscr{P}_3$ (**Agent**$_3$):	go_wedding ←	[**Agent**$_2$]{go_wedding, not drive}
$\mathscr{P}_4$ (**Agent**$_4$):		empty program

Program rules have the form:

$$head \leftarrow [selection\_condition]\{body\},$$

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 agent A will require a in case a number v of agents (other than A) exists such that they require or tolerate b, neither require nor tolerate c and it holds that $0 \leqslant l \leqslant v \leqslant h \leqslant n-1$. By default, $l = 0$ and $h = n-1$. The number n is a parameter—known by each agent—representing the total number of agents (including the agent A). This enriched language is referred to as *social logic programming* (SOLP). The wedding party scenario of Example 1 can be represented by the four SOLP programs shown in Table 1, where the program $\mathscr{P}_4$ is empty since the corresponding agent has not any request or desire to express.

The intended models represent the mental states of each agent inside the community. Concerning the party, such models are the following:

$M_1 = \emptyset$, $M_2 = \{go\_wedding_{\mathscr{P}_1}, go\_wedding_{\mathscr{P}_2}, drive_{\mathscr{P}_2}\}$, and $M_3 = \{go\_wedding_{\mathscr{P}_1}, go\_wedding_{\mathscr{P}_2}, go\_wedding_{\mathscr{P}_3}\}$, where the subscript $_{\mathscr{P}_i}$ $(1 \leqslant i \leqslant n)$ references, for each atom in a model, the program (resp. agent) that atom is entailed by. The models respectively mean that either (M_1) no agent will go to the party, (M_2) only **Agent**$_1$ and **Agent**$_2$ will go and also **Agent**$_2$ will drive the car, or (M_3) all agents but **Agent**$_4$ will go to the party.

Let us show why the above models represent the intended meaning of the program: M_1 is empty in case **Agent**$_2$ does not go to the wedding party (i.e. *go_wedding* is not derived by $\mathscr{P}_2$). Indeed, in such a case, **Agent**$_3$ will not go too, since his requirements w.r.t. **Agent**$_2$ are not satisfied. Moreover, since **Agent**$_4$ expresses neither requirements nor tolerated options, he does not go to the party (observe that such a behavior is also represented by the models M_2 and M_3). Finally, **Agent**$_1$ requires that at least one[2] agent (other than himself) goes to the party. As a consequence of the other agents' behavior, **Agent**$_1$ will not go. Thus, no agent will go to the party and M_1 is empty. The intended meaning of M_2 is that both **Agent**$_1$ and **Agent**$_2$ will go to

[2] Since $\frac{n}{2} - 1 = 1$, where n is the total number of agents, i.e. $n = 4$.

the party and **Agent**$_2$ will also drive the car. In such a case **Agent**$_3$ will not go since he requires that **Agent**$_2$ does not drive the car. The model M_3 represents the case in which **Agent**$_2$ goes to the party, but does not drive the car. Now, since all requirements of **Agent**$_3$ are satisfied, then he will also go to the party. Certainly, **Agent**$_1$ will join the other agents, because, in order to go to the party, he requires that at least one agent goes there.

The intended models are referred to as *social models*, since they express the results of the interactions among agents. As it will be analyzed in Section 6, the multiplicity of intended models is induced both by negation occurring in rule bodies and also directly by the social features, thus making the approach nontrivial.

Let us informally introduce the most important properties of the semantics of the language.

- Social conditions model reasoning conditioned by the behavior of other agents in the community. In particular, it is possible to represent collective mental states, preserving the possibility of identifying the behavior of each agent.
- It is possible to nest social conditions, in order to apply recursively the social-conditioned reasoning to agents' subsets of the community.
- Each social model represents the mental state (i.e. desires, requirements, etc.) of every agent in case the social conditions imposed by the agents are enabled.

Observe that, in order to meet such goals, merging all the input SOLP programs is not enough, since in this way we lose all information about the relationship between an atom and the program (resp. agent) which such an atom comes from. Therefore, we have to find a nontrivial solution.

Our approach starts from (Buccafurri and Gottlob, 2002), where the *joint fixpoint semantics* (JFP), that is a semantics providing a way to reach a compromise (in terms of a common agreement) among agents modeled by logic programs, is proposed. Therein, each model contains atoms representing items being common to all the agents. The approach proposed here in order to reach a social-based conclusion is more general: the agents' behavior is defined by taking into account social conditions specified by the agents themselves.

Informally, a social condition is an expression $[selection\_condition]\{body\}$, where *selection_condition* can be of two forms: either (i) cardinality-based, or (ii) identity-based. In the former case the agent requires that the number of other agents (bounded by *selection_condition* itself) satisfy *body*. In the latter case, *selection_condition* identifies which agent is required to satisfy *body*. Given a program rule including a social condition such as $head \leftarrow [selection\_condition]\{body\}$, the intuitive meaning is that *head* is derived if the social condition is satisfied.

An example of cardinality-based condition [case (i)] is shown in Table 1 by the program $\mathscr{P}_1$: An intended model M will include the atom $go\_wedding_{\mathscr{P}_1}$ if a set of programs $S' \subseteq \{\mathscr{P}_2, \mathscr{P}_3, \mathscr{P}_4\}$ exists such that for each $\mathscr{P} \in S'$, it results that $go\_wedding_{\mathscr{P}}$ belongs to M and also it holds that the number of programs in S' satisfies the social condition imposed by the program $\mathscr{P}_1$, that is $|S'| \geqslant \frac{n}{2} - 1$.

An example of case (ii) (identity-based condition) is represented by the program $\mathscr{P}_3$, which requests the atom $go\_wedding_{\mathscr{P}_3}$ to be part of an intended model M if $go\_wedding_{\mathscr{P}_2}$ belongs to M, but the atom $drive_{\mathscr{P}_2}$ does not. Importantly, social conditions can be nested with each other, as shown by the next example.

Example 2

Consider a peer-to-peer (P2P) file-sharing system where a user can share his collection of files with other users on the Internet. In order to get better performance, a file is split in several parts being downloaded separately (possibly each part from a different user)[3]. The following SOLP program $\mathscr{P}$ describes the behavior of an agent (acting on behalf of a given user) that wants to download every 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 (rule r_1). Moreover, the agent tolerates to share any file X of his, which is shared also by at least 33% of the total number of users in the network and such that among those users, a number of them (bounded between 20% and 70% of the total) exists having a high bandwidth. In this case the agent tolerates to share, since he is sure that the network traffic will not be unbalanced (rule r_2). Observe that the use of nested social conditions in $\mathscr{P}$ is emphasized by means of program indentation.

$$\begin{array}{lrl}
r_1: & download(X) \leftarrow & [min,]\{share(X), \\
 & & \quad [1,]\{not\ incomplete(X) \\
 & & \}, file(X) \\
r_2: & okay(share(X)) \leftarrow & [0.33 * n,]\{share(X), \\
 & & \quad [0.2 * n, 0.7 * n]\{high\_bw\} \\
 & & \}, file(X)
\end{array} .$$

Now, one could argue that a different choice concerning the selection condition could be done. As a first observation we note that the chosen selection conditions play frequently an important role in common-sense reasoning. Indeed, it often happens that the beliefs and the choices of an individual depend on *how many* people think or act in a certain way. For instance, a person who needs a new mobile phone is interested in collecting—from his colleagues or the Internet—a number of opinions on a given model, in order to decide whether he should buy it or not. It occurs also that one is interested in the behavior of a *given* person, in order to act or to infer something. For example, two people are doing shopping together and do not want to buy the same clothes in order not to be dressed in the same way. So, one of them decides not to buy a given item, in case it has been chosen by his partner. These short examples show that two important parameters acting in the social influence are either (i) the number or (ii) the identity of the people involved. For such a reason, we propose a simple, clear-cut, yet general mechanism to represent the selection of a social condition.

As a second observation we remark that this work represents a first step toward a thorough study on how to include in a classical logic-programming setting the paradigm of social interference, in order to directly represent community-based

[3] Among others, KaZaA, EMule, and BitTorrent are the most popular Internet P2P file-sharing systems exploiting such a feature.

reasoning. To this aim, we focus on some suitable selection conditions, but we are aware that other possible choices might be considered. From this perspective, our work tries to give some nontrivial contributions toward what kind of features a knowledge-representation language should include, in order to be oriented to complex scenarios. Anyway, as it will be shown by examples throughout the paper, the chosen social conditions combined with the power of nesting allow us to represent more articulated selections among agents.

Besides the definition of the language, of its semantics, and the application to knowledge representation, another contribution of the paper is the translation of SOLP programs into logic programs with aggregates[4]. In particular, given a set of SOLP programs, a source-to-source transformation exists which provides as output a single logic program with aggregates whose stable models are in one-to-one correspondence with the social models of the set of SOLP programs. The translation to logic programs with aggregates gives us the possibility of exploiting existing engines to compute logic programs.

Moreover, Section 6 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 define the notion of SOLP programs and their semantics (*social semantics*), respectively. In Section 4, we illustrate how a set of SOLP programs, each representing a different agent, is translated into a single logic program with aggregates whose stable models describe the mental states of the whole agent community and then we show that such a translation is sound and complete. In Section 5, we prove that the social semantics extends the JFP semantics (Buccafurri and Gottlob, 2002), and in Section 6 we study the complexity of several interesting decision problems. Section 7 describes how this novel approach may be used for knowledge representation by means of several examples. Then, in Section 8 we discuss related proposals and, finally, we draw our conclusions and sketch the future directions of the work.

In order to improve the overall readability, these sections are followed by Appendix A—where we have placed the proofs of the most complicated technical results—and by the list of symbols and abbreviations used throughout the paper.

2 Syntax of SOLP programs

In this section we first introduce the notion of *social condition* and then we describe the syntax of SOLP programs.

A *term* is either a variable or a constant. Variables are denoted by strings starting with uppercase letters, while those starting with lower case letters denote constants. An *atom* or *positive literal* is an expression $p(t_1,\ldots,t_n)$, where p is a *predicate* of arity n and $t_1,\ldots,t_n$ are terms. A *negative literal* is the *negation as failure (NAF) not a* of a given atom a.

[4] As it is shown in Section 4, we choose the syntax of the nondisjunctive fragment of DLP$^{\mathscr{A}}$ (Dell'Armi *et al.*, 2003), supported by the DLV system (Leone *et al.*, 2002).

Definition 1
Given an integer $n > 0$, an (n-)*social condition* s, also referred to as (n-)SC, is an expression of the form $cond(s)\ property(s)$, such that

(1) $cond(s)$ is an expression $[\alpha]$ where α is either (i) a pair of integers l, h such that $0 \leqslant l \leqslant h \leqslant n-1$ or (ii) a string;
(2) $property(s) = content(s) \cup skel(s)$, where $content(s)$ is a nonempty set of literals and $skel(s)$ is a (possibly empty) set of SCs.

n-Social conditions operate over a collection of n programs representing the agent community. Each agent is modeled by a program (we will formally define later in this section which kind of programs are allowed). n represents the total number of agents. In the following, whenever the context is clear, n is omitted.

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

Concerning item (2) of Definition 1, if $skel(s) = \emptyset$ then s is said *simple.* For a simple SC s such that $content(s)$ is singleton, the enclosing braces can be omitted. Finally, given an SC s, the formula *not s* is referred to as the NAF of s. The following are *simple* SCs extracted from our initial wedding party example (see Table 1):

Example 3
$[\frac{n}{2} - 1,]\{go\_wedding\}$.

Example 4
$[\mathbf{Agent}_2]\{go\_wedding,\ not\ drive\}$.

The social conditions occurring in the example regarding a P2P system (see Example 2) are not simple. As a further example, consider an SC $s = [l, h]\{a, b, c, [l_1, h_1]\{d, [l_2, h_2]e\}, [l_3, h_3]f\}$. Observe that s is not simple, since $skel(s) = \{[l_1, h_1]\{d, [l_2, h_2]e\}, [l_3, h_3]f\}$, moreover, $content(s) = \{a, b, c\}$.

Social conditions enable agents to specify requirements over either individual or groups within the agent community, by using member or cardinal selection conditions, respectively. Moreover, by nesting social conditions it is possible to declare requirements over sub-groups of agents, provided that a super-group satisfying an SC exists. In order to guarantee the correct specifications of nested social conditions, the notion of *well-formed* social condition is introduced next.

Given two n-SCs s and s' such that $cond(s) = [l, h]$ and $cond(s') = [l', h']$ ($0 \leqslant l \leqslant h \leqslant n-1$, $0 \leqslant l' \leqslant h' \leqslant n-1$), if $h' \leqslant h$, then we write $cond(s') \subseteq cond(s)$.

An SC 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 either (a) $cond(s')$ is a cardinal selection condition, s' is a well-formed social condition, and $cond(s') \subseteq cond(s)$, or (b) $cond(s')$ is a member selection condition and s' is simple.

According to the intuitive explanation of the above definition, it results that, besides simple SCs, only nonsimple SCs with cardinal selection condition are candidate to be well formed. Indeed, a nonsimple SC with member selection condition requires some property on a single target agent, but no further sub-group of agents could be specified by means of SCs possibly nested in it. Anyway, a further property is required to SCs with cardinal selection condition in order to be well formed. In particular, given a nonsimple SC s (with cardinal selection condition), all the SCs nested in s with cardinal condition must not exceed the cardinality constraints expressed by $cond(s)$.

Example 5
The SC $s = [1,8]\{a, [3,6]\{b, [\mathbf{Agent_2}]\{c, d\}\}\}$ is well formed. Note that the nonsimple SCs $s_1 = [\mathbf{Agent_3}]\{a, [3,6]\{b\}\}$ and $s_2 = [4,7]\{a, [3,9]b\}$ are not well formed, because $cond(s_1)$ is a member selection condition and, concerning s_2, $[3,9]b \in skel(s_2)$ and $[3,9] \nsubseteq [4,7]$.

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

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

Definition 2
Given an integer $n > 0$, an (n-)*social rule* r is a formula $a \leftarrow b_1 \wedge \cdots \wedge b_m \wedge s_1 \wedge \cdots \wedge s_k$ ($m \geqslant 0$, $k \geqslant 0$), where a is an atom, each b_i ($1 \leqslant i \leqslant m$) is a literal, and each s_j ($1 \leqslant j \leqslant k$) is either an n-SC or the NAF of an n-SC.

Concerning the above definition, the atom a is referred to as the *head* of r, while the conjunction $b_1 \wedge \cdots \wedge b_m \wedge s_1 \wedge \cdots \wedge s_k$ is referred to as the *body* of r.

In case a is of the form $okay(p)$, where p is an atom, then r is referred to as (n-)*tolerance (social) rule*. In case $k = 0$, then a social nontolerance rule is referred to as *classical rule*.

Social tolerance rules, i.e. rules with head of the form $okay(p)$, encode tolerance about the occurrence of p. The rule $okay(p) \leftarrow body$ differs from the rule $p \leftarrow body$ since the latter produces the derivation of p whenever $body$ is satisfied, thus encoding something that is *required* under the condition expressed by $body$. According to the former rule ($okay(p) \leftarrow body$), the truth of $body$ does not necessarily imply p, yet its derivation is not in contrast with the intended meaning of the rule itself. In this sense, under the condition expressed by $body$, p is just *tolerated*.

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

Example 6
An example of nontolerance social rule is $a \leftarrow b, c, [1,9]\{b, c, not\ g, [1,4]\{d\}\}, [\mathscr{P}_2]\{d\}$. An example of tolerance social rule is $okay(a) \leftarrow not\ b, c, [1,6]\{a, not\ f, g\}, not\ [\mathscr{P}_2]\{d\}$.

Definition 3
An SOLP *collection* is a set $\{\mathscr{P}_1, \ldots, \mathscr{P}_n\}$ of SOLP programs, where each SOLP *program* is a set of n-social rules.

An SOLP program is *positive* if no NAF symbol *not* occurs in it. For the sake of presentation we 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 that the reader is familiar with the basic concepts of logic programming (Gelfond and Lifschitz, 1991; Baral, 2003).

We start by introducing the notion of interpretation for a single SOLP program. An *interpretation* for a ground (SOLP)[5] program $\mathscr{P}$ is a subset of $Var(\mathscr{P})$, where $Var(\mathscr{P})$ is the set of atoms appearing in $\mathscr{P}$. 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 *true* w.r.t. I if its head is true or its body is false w.r.t. I.

Recall that, for each traditional logic program Q, the *immediate consequence operator* T_Q is a function from $2^{Var(Q)}$ to $2^{Var(Q)}$ defined as follows. For each interpretation $I \subseteq Var(Q)$, $T_Q(I)$ consists of the set of all heads of rules in Q whose bodies are true w.r.t. I. An interpretation I is a *fixpoint* of a logic program Q if I is a fixpoint of the associated transformation T_Q, i.e., if $T_Q(I) = I$.

The set of all fixpoints of Q is denoted by $FP(Q)$.

Before defining the intended models of our semantics, we need some preliminary definitions.

Let $\mathscr{P}$ be an SOLP program. We define the *autonomous reduction* of $\mathscr{P}$, denoted by $A(\mathscr{P})$, the program obtained from $\mathscr{P}$ by removing all the SCs from the rules in $\mathscr{P}$. The intuitive meaning is that in case the program $\mathscr{P}$ represents the social behavior of an agent, then $A(\mathscr{P})$ represents the behavior of the same agent in case he decides to operate independently of the other agents.

Definition 4
Given an SOLP program $\mathscr{P}$ and an interpretation $I \subseteq Var(A(\mathscr{P}))$, let $TR(A(\mathscr{P}))$ be the set of tolerance rules in $A(\mathscr{P})$ and $Var^*(A(\mathscr{P}))$ be the set $Var(A(\mathscr{P})) \setminus \{okay(p) \mid okay(p) \in Var(A(\mathscr{P}))\} \cup \{p \mid okay(p) \in Var(A(\mathscr{P}))\}$. The *autonomous immediate consequence operator* $AT_{\mathscr{P}}$ is the function from $2^{Var^*(A(\mathscr{P}))}$ to $2^{Var^*(A(\mathscr{P}))}$, defined as follows:

$$\begin{aligned} AT_{\mathscr{P}}(I) = {} & \{head(r) \mid r \in A(\mathscr{P}) \setminus TR(A(\mathscr{P})) \wedge body(r) \text{ is true w.r.t. } I\} \cup \\ & \{a \mid head(r) = okay(a) \wedge r \in TR(A(\mathscr{P})) \wedge (body(r) \wedge a) \text{ is true w.r.t. } I\}. \end{aligned}$$

Observe that $AT_{\mathscr{P}}$, when applied to an interpretation I, extends the classical immediate consequence operator $T_{\mathscr{P}}$, by collecting not only heads of nontolerance rules whose body is true w.r.t. I, but also each atom a occurring as $okay(a)$ in the head of some rule such that both a and the rule body are true w.r.t. I.

Definition 5
An interpretation I for an SOLP program $\mathscr{P}$ is an *autonomous fixpoint* of $\mathscr{P}$ if I is a fixpoint of the associated transformation $AT_{\mathscr{P}}$, i.e. if $AT_{\mathscr{P}}(I) = I$. The set of all autonomous fixpoints of $\mathscr{P}$ is denoted by $AFP(\mathscr{P})$.

[5] We insert SOLP into brackets since the definition is the same as for traditional logic programs.

Observe that by means of the autonomous fixpoints of a given SOLP program $\mathscr{P}$ we represent the mental states of the corresponding agent, assuming that every social condition in $\mathscr{P}$ is not taken into account.

Example 7
Consider the following SOLP program $\mathscr{P}$:

$$\begin{array}{rcl} okay(a) & \leftarrow & b, [1,]\{c\} \\ b & \leftarrow & [2,4]\{d\} \end{array}.$$

It is easy to see that $AFP(\mathscr{P}) = \{\{b\}, \{a, b\}\}$, i.e. the interpretations $I_1 = \{b\}$ and $I_2 = \{a, b\}$ are the autonomous fixpoints of $\mathscr{P}$, since it holds that $AT_{\mathscr{P}}(I_1) = I_1$ and $AT_{\mathscr{P}}(I_2) = I_2$.

Definition 6
Given an SOLP collection $C = \{\mathscr{P}_1, \dots, \mathscr{P}_n\}$, let $\mathscr{P}_i$ $(1 \leqslant i \leqslant n)$ be an SOLP program of C and L be a set of atoms. The *labeled version* of L w.r.t. $\mathscr{P}_i$, denoted by $(L)_{\mathscr{P}_i}$ is the set $\{a_{\mathscr{P}_i} \mid a \in L\}$. Each element of $(L)_{\mathscr{P}_i}$ is referred to as a *labeled atom* w.r.t. $\mathscr{P}_i$.

Example 8
Given an SOLP program $\mathscr{P}_1$ of an SOLP collection C, if $L = \{a, b, c\}$, then $(L)_{\mathscr{P}_1} = \{a_{\mathscr{P}_1}, b_{\mathscr{P}_1}, c_{\mathscr{P}_1}\}$, where the program identifier $\mathscr{P}_1$ indicates the associated SOLP program.

Now we introduce the concept of *social interpretation*, devoted to represent the mental states of the collectivity described by a given SOLP collection and then we give the definition of truth for both literals and SCs w.r.t. a given social interpretation. To this aim, the classical notion of interpretation is extended by means of program identifiers introducing a link between atoms of the interpretation and programs of the SOLP collection.

Definition 7
Let $C = \{\mathscr{P}_1, \dots, \mathscr{P}_n\}$ be an SOLP collection. A *social interpretation* for C is a set $\bar{I} = (I^1)_{\mathscr{P}_1} \cup \cdots \cup (I^n)_{\mathscr{P}_n}$, where I^j is an interpretation for $\mathscr{P}_j$ $(1 \leqslant j \leqslant n)$ and $(I^j)_{\mathscr{P}_j}$ is the labeled version of I^j w.r.t. $\mathscr{P}_j$ (see Definition 6).

Example 9
Given $C = \{\mathscr{P}_1, \mathscr{P}_2, \mathscr{P}_3\}$, $I^1 = \{a, b, c\}$, $I^2 = \{a, d, e\}$, and $I^3 = \{b, c, d\}$, where I^j is an interpretation for $\mathscr{P}_j$ $(1 \leqslant j \leqslant 3)$, then $\bar{I} = \{a_{\mathscr{P}_1}, b_{\mathscr{P}_1}, c_{\mathscr{P}_1}, a_{\mathscr{P}_2}, d_{\mathscr{P}_2}, e_{\mathscr{P}_2}, b_{\mathscr{P}_3}, c_{\mathscr{P}_3}, d_{\mathscr{P}_3}\}$ is a social interpretation for C.

We define now the notion of truth for literals, SCs, and social rules, respectively.

Let $C = \{\mathscr{P}_1, \dots, \mathscr{P}_n\}$ be an SOLP collection. Given a social interpretation $\bar{I}$ for C and a positive literal $a \in \bigcup_{\mathscr{P} \in C} Var(\mathscr{P})$, a (resp. *not* a) is *true* for $\mathscr{P}_j$ $(1 \leqslant j \leqslant n)$ w.r.t. $\bar{I}$ if $a_{\mathscr{P}_j} \in \bar{I}$ (resp. $a_{\mathscr{P}_j} \notin \bar{I}$), otherwise it is *false*.

Because of the recursive nature of SCs, before giving the definition of truth for a SC s, we introduce a way to identify s (and also every SC nested in s) occurring in a given rule r of an SOLP program $\mathscr{P}$. To this aim, we first define a function which returns, for a given SC, its nesting depth.

Given an SC s, we define the function *depth* as follows:

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

Given an SOLP program $\mathscr{P}$, a social rule $r \in \mathscr{P}$ and an integer $n \geqslant 0$, we define the set $MSC^{\langle \mathscr{P},r,n \rangle} = \{s \mid s \text{ is an SC occurring in } r \wedge depth(s) = n\}$, i.e. the set including all the SCs having a given depth n and occurring in a social rule r of an SOLP program $\mathscr{P}$. Observe that, in case the parameter n is zero, then $MSC^{\langle \mathscr{P},r,0 \rangle}$ denotes the set of SCs as they appear in the rule r of $\mathscr{P}$.

Example 10
Let $a \leftarrow [1,8]\{a,[3,6]\{b,[\mathscr{P}_2]\{c,d\}\}\},[2,3]\{e,f\}$ be a rule r in an SOLP program $\mathscr{P}_1$. Then

$$\begin{aligned} MSC^{\langle \mathscr{P},r,0 \rangle} &= \{\ [1,8]\{a,[3,6]\{b,[\mathscr{P}_2]\{c,d\}\}\},[2,3]\{e,f\}\ \}, \\ MSC^{\langle \mathscr{P},r,1 \rangle} &= \{\ [3,6]\{b,[\mathscr{P}_2]\{c,d\}\}\ \}, \\ MSC^{\langle \mathscr{P},r,2 \rangle} &= \{\ [\mathscr{P}_2]\{c,d\}\ \}, \\ MSC^{\langle \mathscr{P},r,3 \rangle} &= \emptyset. \end{aligned}$$

Given an SOLP program $\mathscr{P}$, we define the set $MSC^{\mathscr{P}} = \bigcup_{r \in \mathscr{P}} MSC^{\langle \mathscr{P},r,0 \rangle}$. $MSC^{\mathscr{P}}$ is the set of all the SCs (with depth 0) occurring in $\mathscr{P}$.

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

Definition 8
Let $C = \{\mathscr{P}_1,\ldots,\mathscr{P}_n\}$ be an SOLP collection, $C' \subseteq C$ and $\mathscr{P}_j \in C'$. Given a social interpretation $\bar{I}$ for C and an n-SC $s \in MSC^{\mathscr{P}_j}$, we say that s is *true* for $\mathscr{P}_j$ in C' w.r.t. $\bar{I}$ if it holds that either

(1) $cond(s) = [\mathscr{P}_k]\ \wedge$
$\exists \mathscr{P}_k \in C' \mid \forall a \in content(s),\ a$ is true for $\mathscr{P}_k$ w.r.t. $\bar{I}$, or

(2) $cond(s) = [l,h]\ \wedge$
$\exists D \subseteq C' \setminus \{\mathscr{P}_j\} \mid l \leqslant |D| \leqslant h\ \wedge$
$\forall a \in content(s), \forall \mathscr{P} \in D,\ a$ is true for $\mathscr{P}$ w.r.t. $\bar{I}\ \wedge$
$\forall s' \in skel(s)\ \exists D' \subseteq D \mid s'$ is true for $\mathscr{P}_j$ in D' w.r.t. $\bar{I}$,

where l, h are integers and $\mathscr{P}_k$ is an SOLP program.

If $C' = C$, then we simply say that s is *true* for $\mathscr{P}_j$ w.r.t. $\bar{I}$. An n-SC not true for $\mathscr{P}_j$ (in C') w.r.t. $\bar{I}$ is *false* for $\mathscr{P}_j$ (in C') w.r.t. $\bar{I}$.

Finally, the NAF of an n-SC s, *not* s, is *true* (resp. *false*) for $\mathscr{P}_j$ (in C') w.r.t. $\bar{I}$ if s is false (resp. true) for $\mathscr{P}_j$ (in C') w.r.t. $\bar{I}$.

Informally, given an SC s included in $\mathscr{P}_j$, s is true for $\mathscr{P}_j$ w.r.t. a social interpretation $\bar{I}$ if a single SOLP program $\mathscr{P}_k$ (resp. a set D of SOLP programs not including $\mathscr{P}_j$) exists such that all the elements in $content(s)$ are true for $\mathscr{P}_k$ w.r.t. $\bar{I}$ (resp. for every program $\mathscr{P} \in D$ w.r.t. $\bar{I}$, and such that every element in $skel(s)$ is true for $\mathscr{P}_j$ w.r.t. $\bar{I}$). Observe that the truth of $property(s)$ is possibly defined recursively, since s may contain nested SCs.

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

Let C be an SOLP collection and $\mathscr{P} \in C$. Given a social interpretation $\bar{I}$ for C and a social rule r in $\mathscr{P}$, the head of r is *true* w.r.t. $\bar{I}$ if either (i) $(head(r) = a) \wedge$ (a is true for $\mathscr{P}$ w.r.t. $\bar{I}$), or (ii) $(head(r) = okay(a)) \wedge$ (a is true for $\mathscr{P}$ w.r.t. $\bar{I}$). Moreover, the body of r is *true* w.r.t. $\bar{I}$ if each element of $body(r)$ is true for $\mathscr{P}$ w.r.t. $\bar{I}$. Finally, the social rule r 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 an SOLP collection $\{\mathscr{P}_1, \ldots, \mathscr{P}_n\}$, we define the set of *candidate social interpretations* for $\mathscr{P}_1, \ldots, \mathscr{P}_n$ as

$$\mathscr{U}(\mathscr{P}_1, \ldots, \mathscr{P}_n) = \left\{(F^1)_{\mathscr{P}_1} \cup \cdots \cup (F^n)_{\mathscr{P}_n} \mid F^i \in AFP(\mathscr{P}_i) \wedge 1 \leqslant i \leqslant n\right\},$$

where, recall, $AFP(\mathscr{P}_i)$ is the set of autonomous fixpoints of the SOLP program $\mathscr{P}_i$ (introduced in Definition 5) and by $(F^i)_{\mathscr{P}}$ $(1 \leqslant i \leqslant n)$ we denote the labeled version of F^i w.r.t. $\mathscr{P}$ (see Definition 6).

The set $\mathscr{U}(\mathscr{P}_1, \ldots, \mathscr{P}_n)$ represents all the configurations obtained by combining the autonomous (i.e. without considering the social conditions) mental states of the agents corresponding to the programs $\mathscr{P}_1, \ldots, \mathscr{P}_n$. Each candidate social interpretation is a candidate intended model. The intended models are then obtained by enabling the social conditions.

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

Definition 9
Given an SOLP collection $C = \{\mathscr{P}_1, \ldots, \mathscr{P}_n\}$ and a social interpretation $\bar{I}$ for C, let $\bar{V}$ be the set $(Var(\mathscr{P}_1))_{\mathscr{P}_1} \cup \cdots \cup (Var(\mathscr{P}_n))_{\mathscr{P}_n}$ and $TR(\mathscr{P}_i)$ be the set of tolerance rules of $\mathscr{P}_i$ $(1 \leqslant i \leqslant n)$. The *social immediate consequence operator* ST_C is a function from $2^{\bar{V}}$ to $2^{\bar{V}}$ defined as follows:

$$\begin{aligned} ST_C(\bar{I}) = &\{a_{\mathscr{P}} \mid \mathscr{P} \in C \wedge r \in \mathscr{P} \setminus TR(\mathscr{P}) \wedge head(r) = a \wedge \\ &\ body(r) \text{ is true w.r.t. } \bar{I}\} \cup \\ &\{a_{\mathscr{P}} \mid \mathscr{P} \in C \wedge r \in TR(\mathscr{P}) \wedge head(r) = okay(a) \wedge \\ &\ a \text{ is true for } \mathscr{P} \text{ w.r.t. } \bar{I} \wedge body(r) \text{ is true w.r.t. } \bar{I}\}. \end{aligned}$$

A candidate social interpretation $\bar{I} \in \mathscr{U}(\mathscr{P}_1, \ldots, \mathscr{P}_n)$ for C is a *social model* of C if $ST_C(\bar{I}) = \bar{I}$.

Social models are defined as fixpoints of the operator ST_C. Given a social interpretation $\bar{I}$, $ST_C(\bar{I})$ contains

(1) for each program $\mathscr{P}$ in the SOLP collection C, the labeled versions (w.r.t. $\mathscr{P}$) of the heads of nontolerance rules, such that the body is true w.r.t. $\bar{I}$ (according to Definition 8, all the SCs included in the body are checked w.r.t. the given social interpretation $\bar{I}$).
(2) for each program $\mathscr{P}$ in the SOLP collection C, the labeled versions (w.r.t. $\mathscr{P}$) of the arguments of the predicates *okay* occurring in the heads of tolerance rules, such that both the rule body is true w.r.t. $\bar{I}$ and the predicate argument is true for $\mathscr{P}$ w.r.t. $\bar{I}$.

Observe that the social immediate consequence operator ST_C works differently from the autonomous immediate consequence operator $AT_{\mathscr{P}}$ (see Definition 4), since the former exploits all the programs—and the social conditions included—of a given

SOLP collection C, while the latter operates only within a given program $\mathscr{P}$, where the social conditions have been removed.

Definition 10
Given an SOLP collection $\{\mathscr{P}_1,\ldots,\mathscr{P}_n\}$, the *Social Semantics* of $\mathscr{P}_1,\ldots,\mathscr{P}_n$ is the set $\mathscr{SOS}(\mathscr{P}_1,\ldots,\mathscr{P}_n) = \{\bar{M} \mid \bar{M} \in \mathscr{U}(\mathscr{P}_1,\ldots,\mathscr{P}_n) \wedge \bar{M}$ is a social model of $\mathscr{P}_1,\ldots,\mathscr{P}_n\}$.

$\mathscr{SOS}(\mathscr{P}_1,\ldots,\mathscr{P}_n)$ is the set of all social models of $\mathscr{P}_1,\ldots,\mathscr{P}_n$.

Now, we introduce an important property holding for social models, i.e. they are *supported* in the associated SOLP collection. The next definition gives the notion of supportness for a social model.

Definition 11
Given an SOLP collection $C = \{\mathscr{P}_1,\ldots,\mathscr{P}_n\}$ and a social model $M \in \mathscr{SOS}(\mathscr{P}_1,\ldots,\mathscr{P}_n)$, M is *supported* in C if $\forall \mathscr{P} \in C, \forall a \in Var(\mathscr{P}_1) \cup \cdots \cup Var(\mathscr{P}_n)$, in case $a_{\mathscr{P}} \in M$, then at least one of the following holds:

(1) $\exists r \mid r \in \mathscr{P} \wedge head(r) = a \wedge body(r)$ is true w.r.t. M;
(2) $\exists r \mid r \in \mathscr{P} \wedge head(r) = okay(a) \wedge a$ is true for $\mathscr{P}$ w.r.t. $M \wedge body(r)$ is true w.r.t. M.

The property is stated in the following theorem:

Theorem 1
Given an SOLP collection $C = \{\mathscr{P}_1,\ldots,\mathscr{P}_n\}$, $\forall M \in \mathscr{SOS}(\mathscr{P}_1,\ldots,\mathscr{P}_n)$, M is supported in C.

Proof
By contradiction, assume that $M \in \mathscr{SOS}(\mathscr{P}_1,\ldots,\mathscr{P}_n)$ and M is not supported in C. As a consequence,

$\exists \mathscr{P}, \exists a \mid \mathscr{P} \in C \wedge a \in \bigcup_{\mathscr{P} \in C} Var(\mathscr{P}) \wedge a_{\mathscr{P}} \in M$ and both of the following conditions hold:

(1) $\forall r \in \mathscr{P}$, it holds that $head(r) = a \Rightarrow body(r)$ is false w.r.t. M;
(2) $\forall r \in \mathscr{P}$, it holds that $head(r) = okay(a) \Rightarrow a$ is false for $\mathscr{P}$ w.r.t. $M \wedge body(r)$ is false w.r.t. M.

It is easy to see that, according to Definition 9, $a_{\mathscr{P}} \notin ST_C(M)$. Now, since, according to the hypothesis, $a_{\mathscr{P}} \in M$, it holds that $ST_C(M) \neq M$. Thus M is not a social model and we have reached a contradiction. □

Example 11
Consider the following SOLP collection $C = \{\mathscr{P}_1, \mathscr{P}_2\}$:

$\mathscr{P}_1 : \quad a \leftarrow b, [\mathscr{P}_2]\{c\} \qquad (r_1)$,
$\mathscr{P}_2 : \quad \leftarrow c \qquad (r_2)$.

It holds that $AFP(\mathscr{P}_1) = \{\{a,b\},\emptyset\}$ and $AFP(\mathscr{P}_2) = \{\emptyset\}$. Thus, there exist two candidate social interpretations, namely $I_1 = \{a_{\mathscr{P}_1}, b_{\mathscr{P}_1}\}$ and $I_2 = \emptyset$.

Since both $body(r_1)$ and $body(r_2)$ are false w.r.t. I_1, it holds that $ST_C(I_1) = \emptyset$. As a consequence, I_1 is not a social model of the SOLP collection C. Concerning the social interpretation I_2 it is easy to see that $ST_C(I_2) = \emptyset$. Hence, I_2 is a social model of the SOLP collection C.

Now, consider a slightly different SOLP collection $C' = \{\mathscr{P}'_1, \mathscr{P}'_2\}$

$\mathscr{P}'_1$: $a \leftarrow b, [\mathscr{P}'_2]\{c\}$ (r'_1),
$\mathscr{P}'_2$: $c \leftarrow$ (r'_2).

It holds that $AFP(\mathscr{P}'_1) = \{\{a,b\},\emptyset\}$ and $AFP(\mathscr{P}'_2) = \{c\}$. Thus, we can build the following candidate social interpretations: $I_1 = \{a_{\mathscr{P}'_1}, b_{\mathscr{P}'_1}, c_{\mathscr{P}'_2}\}$ and $I_2 = \{c_{\mathscr{P}'_2}\}$.

Now, since $ST_{C'}(I_1) = \{a_{\mathscr{P}'_1}, c_{\mathscr{P}'_2}\}$ and $\{a_{\mathscr{P}'_1}, c_{\mathscr{P}'_2}\} \neq I_1$, I_1 is not a social model of the collection $C' = \{\mathscr{P}'_1, \mathscr{P}'_2\}$. Finally, $ST_{C'}(I_2) = I_2$, hence I_2 is a social model of C'. It is easy to see that I_2 is supported in C'.

Now, by means of a complete example, we illustrate the notions introduced above.

Example 12
Three agents represented by the SOLP collection $C = \{P_1, P_2, P_3\}$ are as follows:

$$\begin{array}{lrl}
P_1: & go\_party \leftarrow & [2,]\{go\_party, [1,]\{guitar\}\},\\
P_2: & go\_party \leftarrow & [P_3]\{go\_party\}\\
 & guitar \leftarrow & not\ bad\_weather, go\_party,\\
P_3: & go\_party \leftarrow & not\ bad\_weather.
\end{array}$$

The intended meaning of the above SOLP programs is the following: agent P_1 goes to the party only if there are at least other two agents which go there and such that at least one of them brings the guitar with him. Agent P_2 goes to the party only if agent P_3 goes too. Moreover, in case agent P_2 goes and the weather is not bad, then he thinks it is safe to bring the guitar with him. Finally, agent P_3 goes to the party if there is no evidence of bad weather.

It is easy to see that $\mathscr{SOS}(P_1, P_2, P_3) = \{\bar{I}\}$, where $\bar{I}$ is the intended model of the collection C and $\bar{I} = \{go\_party_{P_1}, go\_party_{P_2}, guitar_{P_2}, go\_party_{P_3}\}$.

Indeed, it holds that $AFP(P_1) = \{\{go\_party\}\}$, $AFP(P_2) = \{\{go\_party, guitar\}\}$, and $AFP(P_3) = \{\{go\_party\}\}$. Now, note that the candidate social interpretation $\bar{I}$ is a social model of C, since it holds that $ST_C(\bar{I}) = \bar{I}$. Finally, it is easy to see that $\bar{I}$ is supported in C.

4 Translation to logic programming with aggregates

In this section, we give the translation from SOLP under the social semantics to logic programming with aggregates[6] under the stable model semantics. We assume that the reader is familiar with the stable model semantics (Gelfond and Lifschitz, 1988). Given a traditional logic program $\mathscr{P}$, we denote by $SM(\mathscr{P})$ the set of all the stable models of $\mathscr{P}$. For the sake of presentation, the most complicated proofs are placed in Appendix A.

Our goal is the following: given a collection of SOLP programs we have to generate a single LP$^{\mathscr{A}}$ program whose stable models are in one-to-one correspondence with

[6] We choose the syntax of the nondisjunctive fragment of DLP$^{\mathscr{A}}$ (Dell'Armi *et al.*, 2003), denoted as LP$^{\mathscr{A}}$ in the sequel of the section. The DLV system (Leone *et al.*, 2002) can be used to compute the social models of the SOLP programs.

the social models of the collection. To this aim we perform the following two tasks: (a) we generate a LP$^{\mathscr{A}}$ program by means of a suitable transformation of all the SCs occurring in the SOLP programs of the collection; (b) we obtain another logic program by processing the original SOLP programs in such a way that the SCs are replaced by suitable atoms. Finally, we merge the two programs obtained from tasks (a) and (b) into a single LP$^{\mathscr{A}}$ program. At the end of the section, we present a comprehensive example (Example 13) describing the whole translation process.

Next we describe how task (a) is performed. The first step is the translation of a single SC and the extension of such a translation to all the SCs included in a social rule, an SOLP program, and an SOLP collection, respectively. As a result of task (a), a single LP$^{\mathscr{A}}$ program is generated which represents the translation of the social conditions occurring in the SOLP collection. In order to have fresh literals that allow us to encode—in such a program—the truth of social conditions, we need a mechanism to generate auxiliary atoms that are in one-to-one correspondence with the social conditions occurring in an SOLP program.

Definition 12

Given an SOLP program $\mathscr{P}$, we define $USC^{\mathscr{P}} = \bigcup_{r \in \mathscr{P}} \bigcup_{n \geqslant 0} MSC^{\langle \mathscr{P}, r, n \rangle}$. Moreover, let L^{ρ} and L^{g} be two sets of literals such that both (1) $Var(\mathscr{P})$, L^{ρ}, and L^{g} are disjoint sets and (2) $|L^{\rho}| = |L^{g}| = |USC^{\mathscr{P}}|$. We define two one-to-one mappings, $\rho : USC^{\mathscr{P}} \rightarrow L^{\rho}$ and $g : USC^{\mathscr{P}} \rightarrow L^{g}$.

Observe that, according to the definition of the set $MSC^{\langle \mathscr{P}, r, n \rangle}$ (see p. 11), $USC^{\mathscr{P}}$ is the set of all the SCs (at any nesting depth) in $\mathscr{P}$. Thus, given an SC s included in an SOLP program $\mathscr{P}$, the mapping ρ (resp. g), returns the auxiliary atom $\rho(s)$ (resp. the predicate $g(s)$) such that it is fresh, i.e. it does not occur in $\mathscr{P}$. We will explain next how $\rho(s)$ and $g(s)$ are exploited by the translation process.

The following definition enables the translation of a single social condition s of a given program $\mathscr{P}$ of an SOLP collection C. Observe that this definition is recursive in order to produce the translation of every social condition nested in s. Such a translation produces two sets of rules that we reference as $GUESS^{\mathscr{P}}(s)$ and $CHECK^{\mathscr{P}}(s)$, respectively. Informally, the rules in the set $GUESS^{\mathscr{P}}(s)$ aim at verifying properties concerning atoms belonging to other SOLP programs different from $\mathscr{P}$. These properties are then checked according to the selection condition of s (i.e. $cond(s)$) by means of the rules included in the set $CHECK^{\mathscr{P}}(s)$.

In the definition, $\rho(s)_{\mathscr{P}}$ denotes the atom ρ labeled atom w.r.t. $\mathscr{P}$ (see Definition 6) and it is derived in case the social condition s is true for $\mathscr{P}$ in C w.r.t. a given social interpretation. With a little abuse of notation, $(g(s))(x)_{\mathscr{P}}$ denotes the predicate $g(s)$ labeled w.r.t. $\mathscr{P}$, having argument x.

Definition 13

Given an SOLP collection $SP = \{\mathscr{P}_1, \ldots, \mathscr{P}_n\}$, an integer j $(1 \leqslant j \leqslant n)$, an SOLP program $\mathscr{P}_j \in SP$, and a social condition $s \in USC^{\mathscr{P}_j}$, we define the *SC translation of*

s as the LP$^{\mathscr{A}}$ program $\Psi^{\mathscr{P}_j}(s) = GUESS^{\mathscr{P}_j}(s) \cup CHECK^{\mathscr{P}_j}(s)$, where $GUESS^{\mathscr{P}_j}(s) =$

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

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

$$= \begin{cases} \{\rho(s)_{\mathscr{P}_j} \leftarrow (g(s))(k)_{\mathscr{P}_j}\}, & \text{if } cond(s) = [\mathscr{P}_k] \wedge (1 \leqslant k \leqslant n), \\ \{\rho(s)_{\mathscr{P}_j} \leftarrow \ l \leqslant \#\texttt{count}\{K : (g(s))(K)_{\mathscr{P}_j}, K \neq j\} \leqslant h\} \cup \\ \bigcup_{s' \in skel(s)} CHECK^{\mathscr{P}_j}(s'), & \text{if } cond(s) = [l, h] \end{cases}$$

and `#count` denotes an aggregate function which returns the cardinality of a set of literals satisfying some conditions (Dell'Armi *et al.*, 2003).

The reader may find an instance of application of the above transformation in the final example (Example 13). Now, by means of the next definition, we extend the scope of the above translation to a social rule, an SOLP program and an SOLP collection.

Definition 14
Given an SOLP program $\mathscr{P}$, a social rule $r \in \mathscr{P}$ and an SOLP collection $\{\mathscr{P}_1, \ldots, \mathscr{P}_n\}$, we define

(1) the *SC translation of* r as the LP$^{\mathscr{A}}$ program $T^{\mathscr{P}}(r) = \bigcup_{s \in MSC^{\mathscr{P}}} \Psi^{\mathscr{P}}(s)$;
(2) the *SC translation of* $\mathscr{P}$ as the LP$^{\mathscr{A}}$ program $W^{\mathscr{P}} = \bigcup_{r \in \mathscr{P}} T^{\mathscr{P}}(r)$;
(3) the *SC translation of the collection* as the LP$^{\mathscr{A}}$ program $C(\mathscr{P}_1, \ldots, \mathscr{P}_n) = \bigcup_{1 \leqslant i \leqslant n} W^{\mathscr{P}_i}$.

Observe that given an SOLP program $\mathscr{P}$, for any classical rule $r \in \mathscr{P}$, $T^{\mathscr{P}}(r) = \emptyset$. As a consequence, for any program $\mathscr{P}$ with no social rules, it holds that $W^{\mathscr{P}} = \emptyset$. $C(\mathscr{P}_1, \ldots, \mathscr{P}_n)$ denotes the LP$^{\mathscr{A}}$ program obtained from the processing of all the SCs included in the SOLP collection $\{\mathscr{P}_1, \ldots, \mathscr{P}_n\}$. The generation of $C(\mathscr{P}_1, \ldots, \mathscr{P}_n)$ is the final step of the task (a) within the whole translation machinery.

Now, we describe task (b). We introduce a suitable mapping from SOLP programs to traditional logic programs[7], and then we apply such a transformation to each SOLP program in a given SOLP collection. Finally, we combine the traditional logic programs so obtained into a single program.

Before introducing the mapping, we need a preliminary processing of all tolerance rules in an SOLP program. This is done by means of the following transformation:

Definition 15
Given an SOLP program $\mathscr{P}$, we define the SOLP program $\hat{\mathscr{P}} = \mathscr{P} \setminus TR(\mathscr{P}) \cup \{p \leftarrow p \wedge body(r) \mid r \in TR(\mathscr{P}) \wedge head(r) = okay(p)\}$.

[7] Note that, differently from task (a), the logic program generated here does not contain aggregates.

Note that $\hat{\mathscr{P}}$ is obtained from $\mathscr{P}$ by replacing each tolerance rule $okay(p) \leftarrow body$ with the rule $p \leftarrow p, body$.

The next step gives a mapping from an SOLP program to a traditional logic program.

Definition 16
Let $\mathscr{P}$ be an SOLP program. We define the program $\Gamma'(\hat{\mathscr{P}})$ over the set of atoms $Var(\Gamma'(\hat{\mathscr{P}})) = \{a_{\mathscr{P}} \mid a \in Var(A(\hat{\mathscr{P}}))\} \cup \{a'_{\mathscr{P}} \mid a \in Var(A(\hat{\mathscr{P}}))\} \cup \{sa_{\mathscr{P}} \mid a \in Var(A(\hat{\mathscr{P}}))\} \cup \{fail_{\mathscr{P}}\}$ as $\Gamma'(\hat{\mathscr{P}}) = S'_1(\hat{\mathscr{P}}) \cup S'_2(\hat{\mathscr{P}}) \cup S'_3(\hat{\mathscr{P}})$, where $S'_1(\hat{\mathscr{P}})$, $S'_2(\hat{\mathscr{P}})$, and $S'_3(\hat{\mathscr{P}})$ are defined as follows:

$$S'_1(\hat{\mathscr{P}}) = \quad \{a_{\mathscr{P}} \leftarrow not\ a'_{\mathscr{P}} \mid a \in Var(A(\hat{\mathscr{P}}))\} \cup \{a'_{\mathscr{P}} \leftarrow not\ a_{\mathscr{P}} \mid a \in Var(A(\hat{\mathscr{P}}))\},$$

$$\begin{aligned} S'_2(\hat{\mathscr{P}}) = \quad & \{sa_{\mathscr{P}} \leftarrow b^1_{\mathscr{P}}, \dots, b^n_{\mathscr{P}}, \rho(s_1)_{\mathscr{P}}, \dots, \rho(s_m)_{\mathscr{P}} \mid \\ & a \leftarrow b_1, \dots b_n, s_1, \dots, s_m \in \mathscr{P}\}, \end{aligned}$$

$$\begin{aligned} S'_3(\hat{\mathscr{P}}) = \quad & \{fail_{\mathscr{P}} \leftarrow not\ fail_{\mathscr{P}}, sa_{\mathscr{P}}, not\ a_{\mathscr{P}} \mid a \in Var(A(\hat{\mathscr{P}}))\} \cup \\ & \{fail_{\mathscr{P}} \leftarrow not\ fail_{\mathscr{P}}, a_{\mathscr{P}}, not\ sa_{\mathscr{P}} \mid a \in Var(A(\hat{\mathscr{P}}))\}, \end{aligned}$$

where $A()$ is the autonomous reduction operator (see p. 9).

In other words, given an SOLP program $\mathscr{P}$, first a program $\hat{\mathscr{P}}$ is produced (according to Definition 15) such that all the predicates *okay*() occurring in it are suitably translated. Then, according to Definition 16, three sets of standard logic rules are generated from $\hat{\mathscr{P}}$, referenced as $S'_1(\hat{\mathscr{P}})$, $S'_2(\hat{\mathscr{P}})$, and $S'_3(\hat{\mathscr{P}})$. Observe that atoms occurring in these sets are labeled w.r.t. the source program $\mathscr{P}$ in order not to generate name mismatch in the final merging phase. Informally, the set $S'_1(\hat{\mathscr{P}})$ guesses atoms that are candidates to be included in a social model. By means of the rules included in the set $S'_2(\hat{\mathscr{P}})$, atoms that are supported by a social rule are inferred. The atoms denoted by $\rho(s_i)_{\mathscr{P}}$ $(1 \leqslant i \leqslant m)$ are in one-to-one correspondence with those generated by $W^{\mathscr{P}}$ (see Definition 14) and represent the social conditions occurring in $\mathscr{P}$. Finally, the set $S'_3(\hat{\mathscr{P}})$ ensures that an atom is derived by means of some rule in $S'_2(\hat{\mathscr{P}})$ iff it is also guessed by some rule in $S'_1(\hat{\mathscr{P}})$.

The next definition introduces a logic program representing the translation of the whole SOLP collection.

Definition 17
Given an SOLP collection $\{\mathscr{P}_1, \dots, \mathscr{P}_n\}$, we define the program $P'_u = \bigcup_{1 \leqslant i \leqslant n} \Gamma'(\hat{\mathscr{P}}_i)$.

P'_u is obtained by combining the translations of all the SOLP programs in a given SOLP collection, where the social conditions are replaced by ρ-atoms. The generation of P'_u concludes task (b) of the translation process. Then, the program P'_u is merged with the LP$^{\mathscr{A}}$ program $C(\mathscr{P}_1, \dots, \mathscr{P}_n)$—obtained as a result of task (a)—in order to enable the social conditions [recall that $C(\mathscr{P}_1, \dots, \mathscr{P}_n)$ contains the ρ-atoms as heads of rules, thus allowing the activation of some rule bodies in P'_u]. Finally, the social models of the SOLP collection $\{\mathscr{P}_1, \dots, \mathscr{P}_n\}$ can be found by computing the stable models of the logic program $P'_u \cup C(\mathscr{P}_1, \dots, \mathscr{P}_n)$.

Once we have described how the translation mechanism proceeds, we need to demonstrate that it is sound and complete. To this aim, we have to prove the following results:

(1) The ρ-atoms occurring in $C(\mathscr{P}_1, \dots, \mathscr{P}_n)$ are in one-to-one correspondence with true SCs for $\{\mathscr{P}_1, \dots, \mathscr{P}_n\}$.
(2) A one-to-one correspondence exists between the social models of $\mathscr{P}_1, \dots, \mathscr{P}_n$ and the stable models of the LP$^{\mathscr{A}}$ program $P'_u \cup C(\mathscr{P}_1, \dots, \mathscr{P}_n)$.

First, we prove item (1) given above.

Lemma 1
Given an SOLP collection $SP = \{\mathscr{P}_1, \dots, \mathscr{P}_n\}$, an integer j ($1 \leqslant j \leqslant n$), an SOLP program $\mathscr{P}_j \in SP$, a social interpretation $\bar{I}$ for SP, and an SC $s \in MSC^{\mathscr{P}_j}$, it holds that s is true for $\mathscr{P}_j$ w.r.t. $\bar{I}$ iff $\exists M \in SM(C(\mathscr{P}_1, \dots, \mathscr{P}_n) \cup Q)$ s.t. $\rho(s)_{\mathscr{P}_j} \in M$, where $Q = \{a \leftarrow \mid a \in \bar{I}\}$.

Proof
See Appendix A. □

Intuitively, a given social interpretation $\bar{I}$ will infer rule heads in $C(\mathscr{P}_1, \dots, \mathscr{P}_n)$. These are either labeled ρ-atoms or labeled g-predicates. Lemma 1 states that the ρ-atoms occurring in $C(\mathscr{P}_1, \dots, \mathscr{P}_n)$ are in one-to-one correspondence with true social conditions. Now, since those ρ-atoms occur also in rule bodies of P'_u, in order to replace the corresponding SCs (recall Definitions 16 and 17), they contribute to infer rule heads in P'_u, which represent elements in a social model of the collection $\{\mathscr{P}_1, \dots, \mathscr{P}_n\}$.

Our intention is to compute the social models of $\mathscr{P}_1, \dots, \mathscr{P}_n$ in terms of the stable models of the logic program $P'_u \cup C(\mathscr{P}_1, \dots, \mathscr{P}_n)$[8]. Thus, we must prove item (2). To this aim, let us recall from Buccafurri and Gottlob (2002) some definitions and results that we shall use later.

Definition 18 (Buccafurri and Gottlob 2002)
Let $\mathscr{P}$ be a traditional logic program and $M \subseteq Var(\mathscr{P})$. We denote by $[M]_{\mathscr{P}}$ the set $\{a_{\mathscr{P}} \mid a \in M\} \cup \{a'_{\mathscr{P}} \mid a \in Var(\mathscr{P}) \setminus M\} \cup \{sa_{\mathscr{P}} \mid a \in M\}$.

Definition 19 (Buccafurri and Gottlob 2002)
Let $\mathscr{P}$ be a positive program. We define the program $\Gamma(\mathscr{P})$ over the set of atoms $Var(\Gamma(\mathscr{P})) = \{a_{\mathscr{P}} \mid a \in Var(\mathscr{P})\} \cup \{a'_{\mathscr{P}} \mid a \in Var(\mathscr{P})\} \cup \{sa_{\mathscr{P}} \mid a \in Var(\mathscr{P})\} \cup \{fail_{\mathscr{P}}\}$ as the union of the sets of rules S_1, S_2, and S_3 defined as follows:

$$
\begin{aligned}
S_1 = &\ \{a_{\mathscr{P}} \leftarrow not\ a'_{\mathscr{P}} \mid a \in Var(\mathscr{P})\} \cup \{a'_{\mathscr{P}} \leftarrow not\ a_{\mathscr{P}} \mid a \in Var(\mathscr{P})\} \\
S_2 = &\ \{sa_{\mathscr{P}} \leftarrow b^1_{\mathscr{P}}, \dots, b^n_{\mathscr{P}} \mid a \leftarrow b_1, \dots, b_n \in \mathscr{P}\} \\
S_3 = &\ \{fail_{\mathscr{P}} \leftarrow not\ fail_{\mathscr{P}}, sa_{\mathscr{P}}, not\ a_{\mathscr{P}} \mid a \in Var(\mathscr{P})\} \cup \\
&\ \{fail_{\mathscr{P}} \leftarrow not\ fail_{\mathscr{P}}, a_{\mathscr{P}}, not\ sa_{\mathscr{P}} \mid a \in Var(\mathscr{P})\}.
\end{aligned}
$$

Note that Definition 16 (introducing Γ') can be viewed as an extension of the above definition, since the former takes into account social conditions, while the latter does not. In fact, the transformations Γ' and Γ produce the same result in case of programs with no social conditions.

[8] This can be efficiently accomplished by using the DLV system (Leone *et al.*, 2002).

Proposition 1
Given an SOLP program $\mathscr{P}$, it holds that $\Gamma'(A(\hat{\mathscr{P}})) = \Gamma(A(\hat{\mathscr{P}}))$.

Proof
Since $A(\hat{\mathscr{P}})$ contains no social condition, it is easy to see that, according to Definitions 16 and 4, $S_1'(A(\hat{\mathscr{P}})) = S_1(A(\hat{\mathscr{P}}))$, $S_2'(A(\hat{\mathscr{P}})) = S_2(A(\hat{\mathscr{P}}))$, and $S_3'(A(\hat{\mathscr{P}})) = S_3(A(\hat{\mathscr{P}}))$. As a consequence, $\Gamma'(A(\hat{\mathscr{P}})) = \Gamma(A(\hat{\mathscr{P}}))$. $\square$

Lemma 2 (Buccafurri and Gottlob 2002)
Let $\mathscr{P}$ be a traditional logic program. Then

$$SM(\Gamma(\mathscr{P})) = \bigcup_{F \in FP(\mathscr{P})} \{[F]_{\mathscr{P}}\}.$$

Once we have recalled the results from (Buccafurri and Gottlob, 2002), we introduce some more results that we shall use in order to prove item (2).

Proposition 2
Let $\mathscr{P}$ be an SOLP program. Then $FP(A(\hat{\mathscr{P}})) = AFP(\mathscr{P})$.

Proof
First observe that, given an SOLP program $\mathscr{P}$, it holds that $A(\hat{\mathscr{P}}) = \hat{Q}$ where $Q = A(\mathscr{P})$, i.e. the result of the joint application of the two operators $A()$ and $\hat{}$ is invariant w.r.t. to the order of application. In fact, according to the definitions of both $A()$ (see p. 9) and $\hat{}$ (see Definition 15), it is easy to see that the former operates only on social conditions, while the latter does not, since it operates on both standard atoms and *okay* predicates. Thus, the two operators have disjoint application domains. Hence, the order of application is not relevant.

As a result it holds that, $FP(A(\hat{\mathscr{P}})) = FP(\hat{Q})$ where $Q = A(\mathscr{P})$ and, according to the traditional definition of fixpoint of a logic program (pp. 9), $FP(\hat{Q}) = \{X \mid T_{\hat{Q}}(X) = X \wedge X \in 2^{Var(\hat{Q})}\}$.

Now, according to the definition of the classical immediate consequence operator (pp. 9), $T_{\hat{Q}}(X) = \{head(r) \mid r \in \hat{Q} \wedge body(r)$ is true w.r.t. $X\}$, moreover, according to Definition 15, $\hat{Q} = Q \setminus TR(Q) \cup \{a \leftarrow a, body(r) \mid r \in TR(Q) \wedge \ head(r) = okay(a)\}$. As a consequence,

$$\begin{aligned} &T_{\hat{Q}}(X) \\ &= \{head(r) \mid r \in Q \setminus TR(Q) \wedge body(r) \text{ is true w.r.t. } X\} \cup \\ &\quad \{a \mid head(r) = okay(a) \wedge r \in TR(Q) \wedge (a \wedge body(r)) \text{ is true w.r.t. } X\} \\ &= AT_{\mathscr{P}}(X) \text{ (see Definition 4).} \end{aligned}$$

It is easy to see that $FP(A(\hat{\mathscr{P}})) = FP(\hat{Q}) = \{X \mid T_{\hat{Q}}(X) = X \wedge X \in 2^{Var(\hat{Q})}\}$ $\{X \mid AT_{\mathscr{P}}(X) = X \wedge X \in 2^{Var(A(\hat{\mathscr{P}}))}\}$.

Now, observe that $Var(A(\hat{\mathscr{P}})) = Var^*(A(\mathscr{P}))$ (see Definition 4), since after the application of the operator $\hat{}$ to $A(\mathscr{P})$, each predicate $okay(p)$ is replaced by its argument p, and, according to Definition 4, for each predicate $okay(p)$ appearing in $A(\mathscr{P})$, $okay(p)$ does not occur in $Var^*(A(\mathscr{P}))$, but the argument p does.

As a consequence, it holds that $\{X \mid AT_{\mathscr{P}}(X) = X \wedge X \in 2^{Var(A(\hat{\mathscr{P}}))}\} = \{X \mid AT_{\mathscr{P}}(X) = X \wedge X \in 2^{Var^*(A(\mathscr{P}))}\} = AFP(\mathscr{P})$. $\square$

Now we extend Definition 18 and Lemma 2, given in Buccafurri and Gottlob (2002), to SOLP programs.

Definition 20
Let $\mathscr{P}$ be a (SOLP) program and $M \subseteq Var(\mathscr{P})$. We denote by $[M]_{\mathscr{P}}$ the set $\{a_{\mathscr{P}} \mid a \in M\} \cup \{a'_{\mathscr{P}} \mid a \in Var(A(\hat{\mathscr{P}})) \setminus M\} \cup \{sa_{\mathscr{P}} \mid a \in M\}$.

Given a (SOLP) program $\mathscr{P}$, the operator $[\,]_{\mathscr{P}}$ produces a set of auxiliary atoms labeled w.r.t. $\mathscr{P}$. Those atoms are used in the translation process. Observe that the above definition extends Definition 18, since in case $\mathscr{P}$ is a traditional logic program, then $Var(A(\hat{\mathscr{P}})) = Var(\mathscr{P})$ and thus the two definitions match.

The above results are now exploited in order to prove that, by applying the above operator $[\,]_{\mathscr{P}}$ to the autonomous fixpoints of a given SOLP program $\mathscr{P}$, we obtain the stable models of the translation of the autonomous version of $\mathscr{P}$.

Lemma 3
Given an SOLP program $\mathscr{P}$, it holds that

$$SM(\Gamma'(A(\hat{\mathscr{P}}))) = \bigcup_{F \in AFP(\mathscr{P})} \{[F]_{\mathscr{P}}\}.$$

Proof
By virtue of Proposition 1, $\Gamma'(A(\hat{\mathscr{P}})) = \Gamma(A(\hat{\mathscr{P}}))$. As a consequence, $SM(\Gamma'(A(\hat{\mathscr{P}}))) = SM(\Gamma(A(\hat{\mathscr{P}})))$. Now, denoting $A(\hat{\mathscr{P}})$ by Q, by virtue of Lemma 2, $SM(\Gamma(Q)) = \bigcup_{F \in FP(Q)} \{[F]_Q\}$. According to Definition 18, $[F]_Q = \{a_Q \mid a \in F\} \cup \{a'_Q \mid a \in Var(Q) \setminus F\} \cup \{sa_Q \mid a \in F\}$.

Now, recall that $Q = A(\hat{\mathscr{P}})$. Q represents the SOLP program P, after the application of both $\hat{}$ and $A()$ operators. As a consequence, atoms in $[F]_Q$ are labeled w.r.t. $\mathscr{P}$. Observe that atoms in $\Gamma'(A(\hat{\mathscr{P}}))$ are labeled w.r.t. $\mathscr{P}$ too. Therefore, with a little abuse of notation, we can write $[F]_Q = \{a_{\mathscr{P}} \mid a \in F\} \cup \{a'_{\mathscr{P}} \mid a \in Var(A(\hat{\mathscr{P}})) \setminus F\} \cup \{sa_{\mathscr{P}} \mid a \in F\} = [F]_{\mathscr{P}}$, according to Definition 20 and since Q is a traditional logic program.

Now, we have obtained that $SM(\Gamma(A(\hat{\mathscr{P}}))) = \bigcup_{F \in FP(A(\hat{\mathscr{P}}))} \{[F]_{\mathscr{P}}\}$.

Since, by virtue of Proposition 2, $FP(A(\hat{\mathscr{P}})) = AFP(\mathscr{P})$, it results that

$$\bigcup_{F \in FP(A(\hat{\mathscr{P}}))} \{[F]_{\mathscr{P}}\} = \bigcup_{F \in AFP(\mathscr{P})} \{[F]_{\mathscr{P}}\}. \qquad \square$$

Now we extend Lemma 3 to a whole SOLP collection, but first let us recall the following result from Eiter *et al.* (1997).

Lemma 4 (Eiter et al. 1997)
Let $\mathscr{P} = \mathscr{P}_1 \cup \mathscr{P}_2$ be a program such that $Var(\mathscr{P}_1) \cap Var(\mathscr{P}_2) = \emptyset$. Then

$$SM(\mathscr{P}) = \bigcup_{M_1 \in SM(\mathscr{P}_1), M_2 \in SM(\mathscr{P}_2)} \{M_1 \cup M_2\}.$$

Lemma 5
Given an SOLP collection $\{\mathscr{P}_1, \ldots, \mathscr{P}_n\}$, consider the following sets:

(1) $P_u = \bigcup_{1 \leqslant i \leqslant n} \Gamma'(A(\hat{\mathscr{P}}_i))$,

(2) $SM(P_u)$, and

(3) $$B = \left\{\{(F^1)_{\mathscr{P}_1} \cup (G^1)_{\mathscr{P}_1}\} \bigcup \cdots \bigcup \{(F^n)_{\mathscr{P}_n} \cup (G^n)_{\mathscr{P}_n}\} \mid \forall i\ 1 \leqslant i \leqslant n\ F^i \in AFP(\mathscr{P}_i) \wedge (G^i)_{\mathscr{P}_i} = [F^i]_{\mathscr{P}_i} \setminus (F^i)_{\mathscr{P}_i}\right\}.$$

It holds that $SM(P_u) = B$.

Proof

For each i and j such that $1 \leqslant i \neq j \leqslant n$, according to Definition 16, it holds that $Var(\Gamma'(A(\hat{\mathscr{P}}_i))) \cap Var(\Gamma'(A(\hat{\mathscr{P}}_j))) = \emptyset$. It is easy to see that

$$\begin{aligned} SM(P_u) = {} & SM(\textstyle\bigcup_{1\leqslant i\leqslant n} \Gamma'(A(\hat{\mathscr{P}}_i))) = \text{(by virtue of Lemma 4)} \\ = {} & \bigcup_{M^1 \in SM(\Gamma'(A(\hat{\mathscr{P}}_1))),\ldots,M^n \in SM(\Gamma'(A(\hat{\mathscr{P}}_n)))} \{M^1 \cup \cdots \cup M^n\}. \end{aligned}$$

Note that, for each i $(1 \leqslant i \leqslant n)$, $M^i \in SM(\Gamma'(A(\hat{\mathscr{P}}_i))$ and, by virtue of Lemma 3, $SM(\Gamma'(A(\hat{\mathscr{P}}_i)) = \bigcup_{F \in AFP(\mathscr{P})}\{[F]_{\mathscr{P}}\}$. Thus,

$$\begin{aligned} & \bigcup_{M^1 \in SM(\Gamma'(A(\hat{\mathscr{P}}_1))),\ldots,M^n \in SM(\Gamma'(A(\hat{\mathscr{P}}_n)))} \{M^1 \cup \cdots \cup M^n\} = \\ = {} & \bigcup_{F^1 \in AFP(\mathscr{P}_1),\ldots,F^n \in AFP(\mathscr{P}_n)} \{[F^1]_{\mathscr{P}_1} \cup \cdots \cup [F^n]_{\mathscr{P}_n}\}. \end{aligned}$$

Now, for each i $(1 \leqslant i \leqslant n)$, let us denote by $(G^i)_{\mathscr{P}_i}$ the set $[F^i]_{\mathscr{P}_i} \setminus (F^i)_{\mathscr{P}_i}$. It is easy to see that

$$\begin{aligned} & \bigcup_{F^1 \in AFP(\mathscr{P}_1),\ldots,F^n \in AFP(\mathscr{P}_n)} \{[F^1]_{\mathscr{P}_1} \cup \cdots \cup [F^n]_{\mathscr{P}_n}\} = \\ = {} & \bigcup_{F^1 \in AFP(\mathscr{P}_1),\ldots,F^n \in AFP(\mathscr{P}_n)} \{\{(F^1)_{\mathscr{P}_1} \cup (G^1)_{\mathscr{P}_1}\} \cup \cdots \cup \{(F^n)_{\mathscr{P}_n} \cup (G^n)_{\mathscr{P}_n}\}\} = \\ = {} & B. \end{aligned}$$ □

Before proving (2) we need a further definition, introducing the notion of a set of ρ-atoms and g-predicates associated, by virtue of Lemma 1, with the social conditions true for a given SOLP program w.r.t. a social interpretation.

Definition 21

Given an SOLP collection $SP = \{\mathscr{P}_1, \ldots, \mathscr{P}_n\}$, a social interpretation $\bar{I}$ for SP, an SOLP program $\mathscr{P} \in SP$ and an SC $s \in MSC^{\mathscr{P}}$, let $Q = \{a \leftarrow \mid a \in \bar{I}\}$. We define the set

$$SAT_{\bar{I}}^{\mathscr{P}}(s) = \left\{h \mid h = head(r), r \in \Psi^{\mathscr{P}}(s) \wedge (\exists M \in SM(C(\mathscr{P}_1, \ldots, \mathscr{P}_n) \cup Q) \mid h \in M)\right\}.$$

Observe that in case s is true for $\mathscr{P}$ w.r.t. $\bar{I}$, $SAT_{\bar{I}}^{\mathscr{P}}(s)$ includes the atom $\rho(s)_{\mathscr{P}}$ and those heads of the rules in $\Psi^{\mathscr{P}}(s)$ [recall from Definition 13 that $\Psi^{\mathscr{P}}(s) = GUESS^{\mathscr{P}}(s) \cup CHECK^{\mathscr{P}}(s)$] corresponding to both the social condition s and the SCs nested in s.

Finally, we are ready to prove item (2). The next theorem states that a one-to-one correspondence exists between the social models in $\mathscr{SOS}(\mathscr{P}_1, \ldots, \mathscr{P}_n)$ and the stable models of the LP$^{\mathscr{A}}$ program $P'_u \cup C(\mathscr{P}_1, \ldots, \mathscr{P}_n)$.

Theorem 2
Given an SOLP collection $SP = \{\mathscr{P}_1, \ldots, \mathscr{P}_n\}$, it holds that $A = B$, where

$$
\begin{array}{rll}
A = & SM(P'_u \cup C(\mathscr{P}_1, \ldots, \mathscr{P}_n)) & \text{and} \\
B = & \{\bar{F} \cup \bar{G} \cup \bar{H} \mid & \\
(1) & \bar{F} = \bigcup_{1 \leqslant i \leqslant n} (F^i)_{\mathscr{P}_i} \wedge F^i \in AFP(\mathscr{P}_i) \wedge \bar{F} \in \mathscr{SOS}(\mathscr{P}_1, \ldots, \mathscr{P}_n) \wedge & \\
(2) & \bar{G} = \bigcup_{1 \leqslant i \leqslant n} (G^i)_{\mathscr{P}_i} \wedge (G^i)_{\mathscr{P}_i} = [F^i]_{\mathscr{P}_i} \setminus (F^i)_{\mathscr{P}_i} \wedge & \\
(3) & \bar{H} = \bigcup_{1 \leqslant i \leqslant n} (H^i)_{\mathscr{P}_i} \wedge (H^i)_{\mathscr{P}_i} = \bigcup_{s \in MSC^{\mathscr{P}_i}} SAT_{\bar{F}}^{\mathscr{P}_i}(s)\}. &
\end{array}
$$

Proof
See Appendix A.

□

As a result of the above theorem, each stable model X of the program $P'_u \cup C(\mathscr{P}_1, \ldots, \mathscr{P}_n)$ may be partitioned in three sets: $\bar{F}$ (representing the corresponding social model of the SOLP collection), $\bar{G}$ and $\bar{H}$ (each including auxiliary literals needed by the translation). Thus, it is possible to find the social models of $\mathscr{P}_1, \ldots, \mathscr{P}_n$ by a post-processing of the stable models of $P'_u \cup C(\mathscr{P}_1, \ldots, \mathscr{P}_n)$, which drops the sets $\bar{G}$ and $\bar{H}$.

Example 13
Before closing the section, we present the following logic program $\mathscr{P} = P'_u \cup C(\mathscr{P}_1, \ldots, \mathscr{P}_4)$ resulting from the translation of the SOLP collection $\{\mathscr{P}_1, \ldots, \mathscr{P}_4\}$ presented in Example 1 (see Table 1).

$$
\begin{array}{rrcl}
r_1: & go\_wedding_{P_1} & \leftarrow & not\ go\_wedding'_{P_1} \\
r_2: & go\_wedding'_{P_1} & \leftarrow & not\ go\_wedding_{P_1} \\
r_3: & sgo\_wedding_{P_1} & \leftarrow & \rho\_1\_1_{P_1} \\
r_4: & fail_{P_1} & \leftarrow & not\ fail_{P_1}, sgo\_wedding_{P_1}, not\ go\_wedding_{P_1} \\
r_5: & fail_{P_1} & \leftarrow & not\ fail_{P_1}, go\_wedding_{P_1}, not\ sgo\_wedding_{P_1} \\
r_6: & go\_wedding_{P_2} & \leftarrow & not\ go\_wedding'_{P_2} \\
r_7: & go\_wedding'_{P_2} & \leftarrow & not\ go\_wedding_{P_2} \\
r_8: & drive_{P_2} & \leftarrow & not\ drive'_{P_2} \\
r_9: & drive'_{P_2} & \leftarrow & not\ drive_{P_2} \\
r_{10}: & sgo\_wedding_{P_2} & \leftarrow & go\_wedding_{P_2} \\
r_{11}: & sdrive_{P_2} & \leftarrow & drive_{P_2}, go\_wedding_{P_2} \\
r_{12}: & fail_{P_2} & \leftarrow & not\ fail_{P_2}, sgo\_wedding_{P_2}, not\ go\_wedding_{P_2} \\
r_{13}: & fail_{P_2} & \leftarrow & not\ fail_{P_2}, go\_wedding_{P_2}, not\ sgo\_wedding_{P_2} \\
r_{14}: & fail_{P_2} & \leftarrow & not\ fail_{P_2}, sdrive_{P_2}, not\ drive_{P_2} \\
r_{15}: & fail_{P_2} & \leftarrow & not\ fail_{P_2}, drive_{P_2}, not\ sdrive_{P_2} \\
r_{16}: & go\_wedding_{P_3} & \leftarrow & not\ go\_wedding'_{P_3} \\
r_{17}: & go\_wedding'_{P_3} & \leftarrow & not\ go\_wedding_{P_3} \\
r_{18}: & sgo\_wedding_{P_3} & \leftarrow & \rho\_1\_1_{P_3} \\
r_{19}: & fail_{P_3} & \leftarrow & not\ fail_{P_3}, sgo\_wedding_{P_3}, not\ go\_wedding_{P_3} \\
r_{20}: & fail_{P_3} & \leftarrow & not\ fail_{P_3}, go\_wedding_{P_3}, not\ sgo\_wedding_{P_3} \\
r_{21}: & g\_1\_1(2)_{P_1} & \leftarrow & go\_wedding_{P_2} \\
r_{22}: & g\_1\_1(3)_{P_1} & \leftarrow & go\_wedding_{P_3} \\
r_{23}: & \rho\_1\_1_{P_1} & \leftarrow & 1 <= \#count\{K : g\_1\_1(K)_{P_1}, K <> 1\} <= 3 \\
r_{24}: & g\_1\_1(2)_{P_3} & \leftarrow & go\_wedding_{P_2}, not\ drive_{P_2} \\
r_{25}: & \rho\_1\_1_{P_3} & \leftarrow & g\_1\_1(2)_{P_3}
\end{array}
$$

Observe that, according to Definition 16, $\Gamma'(\hat{\mathscr{P}}_1) = S'_1(\hat{\mathscr{P}}_1) \cup S'_2(\hat{\mathscr{P}}_1) \cup S'_3(\hat{\mathscr{P}}_1)$, where $S'_1(\hat{\mathscr{P}}_1) = \{r_1, r_2\}$, $S'_2(\hat{\mathscr{P}}_1) = \{r_3\}$, and $S'_3(\hat{\mathscr{P}}_1) = \{r_4, r_5\}$. Concerning the SOLP program $\mathscr{P}_2$, $\Gamma'(\hat{\mathscr{P}}_2) = S'_1(\hat{\mathscr{P}}_2) \cup S'_2(\hat{\mathscr{P}}_2) \cup S'_3(\hat{\mathscr{P}}_2)$, where $S'_1(\hat{\mathscr{P}}_2) = \{r_6, r_7, r_8, r_9\}$, $S'_2(\hat{\mathscr{P}}_2) = \{r_{10}, r_{11}\}$, and $S'_3(\hat{\mathscr{P}}_2) = \{r_{12}, r_{13}, r_{14}, r_{15}\}$. $\Gamma'(\hat{\mathscr{P}}_3) = S'_1(\hat{\mathscr{P}}_3) \cup S'_2(\hat{\mathscr{P}}_3) \cup S'_3(\hat{\mathscr{P}}_3)$, where $S'_1(\hat{\mathscr{P}}_3) = \{r_{16}, r_{17}\}$, $S'_2(\hat{\mathscr{P}}_3) = \{r_{18}\}$, and $S'_3(\hat{\mathscr{P}}_3) = \{r_{19}, r_{20}\}$. Finally, $\Gamma'(\hat{\mathscr{P}}_4) = \emptyset$ since $\mathscr{P}_4$ is empty. Recall that $P'_u = \Gamma'(\hat{\mathscr{P}}_1) \cup \Gamma'(\hat{\mathscr{P}}_2) \cup \Gamma'(\hat{\mathscr{P}}_3) \cup \Gamma'(\hat{\mathscr{P}}_4)$ (see Definition 17).

Now, according to Definition 14, $C(\mathscr{P}_1, \ldots, \mathscr{P}_4) = \{W^{\mathscr{P}_1} \cup W^{\mathscr{P}_2} \cup W^{\mathscr{P}_3} \cup W^{\mathscr{P}_4}\}$, where $W^{\mathscr{P}_1} = \{r_{21}, r_{22}, r_{23}\}$, $W^{\mathscr{P}_2}$ and $W^{\mathscr{P}_4}$ are empty (since $\mathscr{P}_2$ and $\mathscr{P}_4$ do not include any social rule) and, finally, $W^{\mathscr{P}_3} = \{r_{24}, r_{25}\}$. It is easy to check that the stable models of $\mathscr{P}$ correspond, by virtue of Theorem 2, to the social models of the SOLP programs $\mathscr{P}_1, \ldots, \mathscr{P}_4$.

5 Social models and joint fixpoints

In this section, we show that the social semantics extends the JFP semantics (Buccafurri and Gottlob, 2002). Basically, COLP programs are logic programs which also contain *tolerance* rules (named *okay* rules) which are rules of the form $okay(p) \leftarrow body(r)$. The semantics of a collection of COLP programs is defined over traditional programs obtained from 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 $\mathscr{P}_1, \ldots, \mathscr{P}_n$ of COLP programs is defined by Buccafurri and Gottlob (2002) in terms of joint (i.e., common) fixpoints (of the *immediate consequence operator*) of the logic programs obtained from $\mathscr{P}_1, \ldots, \mathscr{P}_n$ by transforming *okay* rules occurring in them (as shown above).

First we need some preliminary definitions and results. We define a translation from COLP programs Buccafurri and Gottlob (2002) to SOLP programs.

Definition 22

Given a COLP program $\mathscr{P}$ and an integer $n \geqslant 1$, the *SOLP translation* of $\mathscr{P}$ is an SOLP program

$$\sigma^n(\mathscr{P}) = \{\sigma^n(r) \mid r \in \mathscr{P}\},$$

where

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

Definition 23

Given a COLP program $\mathscr{P}$, let $OKAY(\mathscr{P})$ be the set of all the *okay* rules included in $\mathscr{P}$. We define $\hat{\mathscr{P}} = \mathscr{P} \setminus OKAY(\mathscr{P}) \cup \{p \leftarrow p, body(r) \mid r \in OKAY(\mathscr{P}) \wedge head(r) = okay(p)\}$.

Informally, for any given COLP program $\mathscr{P}$, $\hat{\mathscr{P}}$ is a traditional logic program obtained from $\mathscr{P}$ by replacing the head of each *okay* rule with the argument of the predicate *okay* and then adding such an argument to the body of the rule.

Lemma 6
Given a COLP program $\mathscr{P}$, then

$$\forall n \geqslant 1 \quad FP(\hat{\mathscr{P}}) = AFP(\sigma^n(\mathscr{P})).$$

Proof
First, observe that, according to Definition 4, all SCs occurring in the program $\sigma^n(\hat{\mathscr{P}})$ are discarded in order to compute the autonomous fixpoints. As a consequence, the equivalence holds for any value of the parameter n. Now, it is easy to see that the proof follows directly from Definitions 5, 22, and 23. □

Lemma 7
Let $\mathscr{P}_1,\dots,\mathscr{P}_n$ be a set of COLP programs and $Q_1,\dots,Q_n$ be SOLP programs such that $Q_i = \sigma^n(\mathscr{P}_i)$. Then

$$\mathscr{SOS}(Q_1,\dots,Q_n) = \{(F^1)_{Q_1} \cup \cdots \cup (F^n)_{Q_n} \mid \forall i,j \quad 1 \leqslant i \neq j \leqslant n, \quad F^i = F^j\}.$$

Proof
By contradiction let us assume that

$$\begin{array}{l}(1)\ \exists \bar{M} \mid \bar{M} = (F^1)_{Q_1} \cup \cdots \cup (F^n)_{Q_n} \wedge \bar{M} \in \mathscr{SOS}(Q_1,\dots,Q_n), \text{ and} \\ (2)\ \exists i,j \quad 1 \leqslant i \neq j \leqslant n \mid F^i \neq F^j.\end{array}$$

Thus, without loss of generality there exists $h \in F^i$ s.t. $h_{Q_i} \in (F^i)_{Q_i}$ and $h_{Q_j} \notin (F^j)_{Q_j}$. As a consequence, $h_{Q_i} \in \bar{M}$ and $h_{Q_j} \notin \bar{M}$. Now, we have reached a contradiction because, according to Definitions 10 and 22, $h_{Q_i} \in \bar{M}$ only if for each k, $(1 \leqslant k \neq i \leqslant n)$, $h_{Q_k} \in \bar{M}$, thus $\bar{M} \notin \mathscr{SOS}(Q_1,\dots,Q_n)$ (contradiction). □

The next theorem states that the JFP semantics is a special case of the social semantics. $JFP(\mathscr{P}_1,\dots,\mathscr{P}_n)$ denotes the set of the joint fixpoints of the collection of COLP programs $\mathscr{P}_1,\dots,\mathscr{P}_n$.

Theorem 3
Let $\mathscr{P}_1,\dots,\mathscr{P}_n$ $(n \geqslant 1)$ be COLP programs and $C = \{Q_1,\dots,Q_n\}$ be a collection of SOLP programs such that $Q_i = \sigma^n(\mathscr{P}_i)$. Then

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

Proof
$(\supseteq)$. First, we show that

$$\forall F \in JFP(\mathscr{P}_1,\dots,\mathscr{P}_n), \quad \bigcup_{1 \leqslant i \leqslant n} (F)_{Q_i} \in \mathscr{SOS}(Q_1,\dots,Q_n).$$

By contradiction, let us assume that $\exists \bar{M} = \bigcup_{1 \leqslant i \leqslant n} (F)_{Q_i} \mid \bar{M} \notin \mathscr{SOS}(Q_1,\dots,Q_n)$. Thus, there exists an integer j, $1 \leqslant j \leqslant n$, such that either

(1) $F \notin AFP(Q_j)$, or
(2) $F \in AFP(Q_j) \wedge ST_C(\bar{M}) \neq \bar{M}$.

In case (1), by virtue of Lemma 6, $F \notin FP(\hat{P}_j)$ which contradicts the hypothesis that $F \in JFP(\mathscr{P}_1,\dots,\mathscr{P}_n)$.

In case (2), it holds that either

(a) $\exists h, Q_j \mid Q_j \in C \wedge h_{Q_j} \in \bar{M} \wedge h_{Q_j} \notin ST_C(\bar{M})$, or
(b) $\exists h, Q_j \mid Q_j \in C \wedge h_{Q_j} \notin \bar{M} \wedge h_{Q_j} \in ST_C(\bar{M})$.

If condition (a) occurs, then, for each rule $r \in Q_j$, s.t. $head(r) = h$ or $head(r) = okay(h)$, it results that $body(r)$ is false w.r.t. $\bar{M}$, because $h_{Q_j} \notin ST_C(\bar{M}$ (recall Definition 9).

Now, since $h \in F$ and $F \in AFP(Q_j)$, according to Definition 22, it holds that for each rule $r \in Q_j$, s.t. $head(r) = h$ or $head(r) = okay(h)$, the SC $[n-1, n-1]h$ (introduced by the transformation σ) is false for Q_j w.r.t. $\bar{M}$.

Thus, according to Definition 8, there exists k ($1 \leqslant k \neq j \leqslant n$) s.t. $h_{Q_k} \notin \bar{M}$. Now, we have obtained that $h_{Q_j} \in \bar{M}$ and $h_{Q_k} \notin \bar{M}$. Since $\bar{M} = \bigcup_{1 \leqslant i \leqslant n}(F)_{Q_i}$ and $F \in JFP(\mathscr{P}_1, \ldots, \mathscr{P}_n)$, by virtue of Lemma 7, we have reached a contradiction. This concludes the proof of case (2), when condition (a) holds.

Consider, now, that condition (b) is true for case (2). According to Definition 9, there exists $r \in Q_j$ s.t. $head(r) = h$ or $head(r) = okay(h)$ and $body(r)$ is true w.r.t. $\bar{M}$. As a consequence and according to Definition 22, $[n-1, n-1]h$ is true for Q_j w.r.t. $\bar{M}$. Now, according to Definition 8, for each k ($1 \leqslant k \neq j \leqslant n$), $h_{Q_k} \in \bar{M}$ and $h_{Q_j} \notin \bar{M}$. Since $\bar{M} = \bigcup_{1 \leqslant i \leqslant n}(F)_{Q_i}$ and $F \in JFP(\mathscr{P}_1, \ldots, \mathscr{P}_n)$, by virtue of Lemma 7, we have reached a contradiction. This concludes the proof of case (2).

($\subseteq$). Now we show that $\forall \bar{M} \in \mathscr{SOS}(Q_1, \ldots, Q_n)$, it holds that both

(1) $\bar{M} = \{\bigcup_{1 \leqslant i \leqslant n}(F^i)_{Q_i} \mid \forall i, j \ (1 \leqslant i \neq j \leqslant n), F^i = F^j\}$ and
(2) $\forall i \ (1 \leqslant i \leqslant n),\ F^i \in JFP(\mathscr{P}_1, \ldots, \mathscr{P}_n)$.

First observe that condition (1) follows directly from Lemma 7.

Now we prove that condition (2) is true. Assume, by contradiction, that $F \notin JFP(\mathscr{P}_1, \ldots, \mathscr{P}_n)$, where $F = F^1 = F^2 = \cdots = F^n$ (thanks to condition (1)). As a consequence, there exists j ($1 \leqslant j \leqslant n$) s.t. $F \notin FP(\hat{\mathscr{P}}_j)$ (recall that $\mathscr{P}_j$ is a COLP program). Now, by virtue of Lemma 6, $F \notin AFP(Q_j)$, where $Q_j = \sigma^n(\mathscr{P}_j)$. Thus, according to Definition 10, $F_{Q_j} \notin \bar{M}$, because F_{Q_j} is not an autonomous fixpoint of Q_j. This result contradicts condition (1), which is true, stating that $F_{Q_j} \subseteq \bar{M}$. □

6 Complexity results

In this section, we introduce some relevant decision problems with respect to the social semantics and discuss their complexity. The analysis is done in case of positive programs. The extension to the general case is straightforward.

First, we consider the problem of social model existence for a collection of SOLP programs.

PROBLEM SOS_n (social model existence):

Instance: An SOLP collection $C = \{\mathscr{P}_1, \ldots, \mathscr{P}_n\}$.
Question: Is $\mathscr{SOS}(\mathscr{P}_1, \ldots, \mathscr{P}_n) \neq \emptyset$, i.e., do the programs $\mathscr{P}_1, \ldots, \mathscr{P}_n$ have any social model?

Theorem 4
The problem SOS_n is NP-complete.

Proof
(1) Membership. In order to verify that a set of positive SOLP programs admits a social model, it suffices to guess a candidate social interpretation $\bar{I}$ for C and then to check that $ST_C(\bar{I}) = \bar{I}$. Since the latter task is feasible in polynomial time, then the problem SOS_n is in NP.

(2) Hardness. Observe that the problem SOS_n generalizes the problem JFP (Buccafurri and Gottlob, 2002), which has been proved to NP-complete. Indeed, in Definition 22 a polynomial-time reduction from JFP to SOS_n, i.e. $\sigma^n(\mathscr{P})$, has been introduced. Moreover, Theorem 3 states that any instance of the problem JFP can be reduced to an equivalent instance of SOS_n, i.e. on those instances the both problems have the same answers. Thus, we have proven that the problem SOS_n is NP-hard. Then the problem SOS_n is NP-complete. □

Indeed, the case of nonpositive programs is straightforward: since it is NP-complete to determine whether a *single* nonpositive program has a fixpoint, it is easy to see that the same holds for nonpositive SOLP programs and autonomous fixpoints. Thus, checking whether an SOLP collection containing at least one nonpositive 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.

Now, we introduce several computationally interesting decision problems associated with the social semantics. Each of them corresponds to a computational task involving labeled atom search inside the social models of an SOLP collection.

The traditional approach used for classical nonmonotonic semantics of logic programs, typically addresses the following two problems:

(a) *Skeptical reasoning*, i.e. deciding whether an atom x occurs in all the models of a given program $\mathscr{P}$;
(b) *Credulous reasoning*, i.e. deciding whether an atom x occurs in some model of a given program $\mathscr{P}$.

Since a social model is a social interpretation, i.e. a set of labeled atoms, we have to extend the above problems (a) and (b) by introducing a further search dimension, expressing the *sociality* degree of the agents represented by the SOLP collection. More informally, we are also interested in *how many* SOLP programs a given atom—occurring as a labeled atom in a social model—is entailed by.

As a consequence, given a collection $C = \{\mathscr{P}_1, \ldots, \mathscr{P}_n\}$ of SOLP programs, a social model $\bar{M}$ of $\mathscr{P}_1, \ldots, \mathscr{P}_n$ and an atom x, we distinguish two cases. Either

(1) for each $\mathscr{P}$ in C, $\forall i$ $(1 \leqslant i \leqslant n)$, $x_{\mathscr{P}_i} \in \bar{M}$, or
(2) for some $\mathscr{P}$ in C, $\exists i \mid 1 \leqslant i \leqslant n \wedge x_{\mathscr{P}_i} \in \bar{M}$.

In other words, in case (1) the agents corresponding to the SOLP collection C exhibit a greater sociality degree—since all of them choose the atom x inside the social model $\bar{M}$—than in case (2), where at least one agent is required to choose x and thus we observe a more individual agent behavior.

By combining the problems (a) and (b) with the traditional reasoning tasks (1) and (2), we obtain the following four decision problems relevant to the social semantics:

1. PROBLEM $SS\text{-}SOS_n$ (socially skeptical reasoning):
Instance: An SOLP collection $C = \{\mathscr{P}_1, \ldots, \mathscr{P}_n\}$ and an atom x.
Question: Does it hold that, for each $\bar{M} \in \mathscr{SOS}(\mathscr{P}_1, \ldots, \mathscr{P}_n)$, $\{x_{\mathscr{P}_1}, \ldots, x_{\mathscr{P}_n}\} \subseteq \bar{M}$, i.e. $\forall i\ (1 \leqslant i \leqslant n), x_{\mathscr{P}_i} \in \bar{M}$?

In case the answer to such a problem is positive, then it holds that all the agents always (i.e., in each social model) choose x, since $x_{\mathscr{P}}$ occurs in $\bar{M}$, for each social model $\bar{M}$ and for each SOLP program $\mathscr{P}$. For instance, this kind of reasoning could be applied by the government of a given country in order to know if all citizens, modeled as a collection of SOLP programs, pay taxes.

2. PROBLEM $IS\text{-}SOS_n$ (individually skeptical reasoning):
Instance: An SOLP collection $C = \{\mathscr{P}_1, \ldots, \mathscr{P}_n\}$ and an atom x.
Question: Does it hold that, for each $\bar{M} \in \mathscr{SOS}(\mathscr{P}_1, \ldots, \mathscr{P}_n)$, $\exists i \mid 1 \leqslant i \leqslant n \wedge x_{\mathscr{P}_i} \in \bar{M}$?

In case the answer to such a problem is positive, then it holds that always (i.e., in every social model) there is at least an agent choosing x, since $x_{\mathscr{P}}$ occurs in $\bar{M}$, for each social model $\bar{M}$ and for some SOLP program $\mathscr{P}$. This kind of reasoning is useful, for instance, to test if a given action, represented by x, is always performed by at least one agent, no matter who the agent is. For example, consider a family (modeled as an SOLP collection) sharing a car. The above kind of reasoning could be used in order to check whether someone gets gasoline each time the car is used.

3. PROBLEM $SC\text{-}SOS_n$ (socially credulous reasoning):
Instance: An SOLP collection $C = \{\mathscr{P}_1, \ldots, \mathscr{P}_n\}$ and an atom x.
Question: Does it hold that, there exists $\bar{M} \in \mathscr{SOS}(\mathscr{P}_1, \ldots, \mathscr{P}_n)$, such that for each $i\ (1 \leqslant i \leqslant n)$, $x_{\mathscr{P}_i} \in \bar{M}$, i.e. $\{x_{\mathscr{P}_1}, \ldots, x_{\mathscr{P}_n}\} \subseteq \bar{M}$?

In case the answer to such a problem is positive, then at least one social model exists whereas all the agents choose x, since $x_{\mathscr{P}}$ occurs in $\bar{M}$, for some social model $\bar{M}$ and for each SOLP program $\mathscr{P}$. As a consequence, a common agreement on x by the agents may be reached at least in one case (i.e. in one social model). For instance, this kind of reasoning could be applied in order to check whether some chance exists that the European Council of Ministers (modeled as an SOLP collection) unanimously accepts a country as a new member of the European Union.

4. PROBLEM $IC\text{-}SOS_n$ (individually credulous reasoning):
Instance: An SOLP collection $C = \{\mathscr{P}_1, \ldots, \mathscr{P}_n\}$ and an atom x.
Question: Does it hold that, there exist $\bar{M}$ and j such that (i) $\bar{M} \in \mathscr{SOS}(\mathscr{P}_1, \ldots, \mathscr{P}_n)$ and (ii) it holds that $1 \leqslant j \leqslant n \wedge x_{\mathscr{P}_j} \in \bar{M}$?

In case the answer to such a problem is positive, then it holds that at least one social model exists whereas at least one agent chooses x, since $x_{\mathscr{P}}$ occurs in $\bar{M}$, for some social model $\bar{M}$ and for some SOLP program $\mathscr{P}$. In such a case, although there is no common agreement on x by the agents, it holds that x is chosen by some of them, at least once (i.e. in one social model). For example, the above reasoning could be used by a company in order to check whether a given product is never bought by a group of potential customers (represented by SOLP programs).

The computational complexity of the above problems is stated by the following theorems.

Theorem 5
The problem $SS\text{-}SOS_n$ is coNP-complete.

Proof
It suffices to prove that the complementary problem of $SS\text{-}SOS_n$ is NP-complete. Such a problem may be described as follows:

Instance: An SOLP collection $C = \{\mathscr{P}_1, \ldots, \mathscr{P}_n\}$ and an atom x.
Question: Does it hold that there exist $\bar{M}$ and j such that (i) $\bar{M} \in \mathscr{SOS}(\mathscr{P}_1, \ldots, \mathscr{P}_n)$ and (ii) $1 \leqslant j \leqslant n \wedge x_{\mathscr{P}_j} \notin \bar{M}$?

(1) Membership. We need to guess a candidate social interpretation $\bar{M}$ for C and, then, to verify that

(i) $\bar{M} \in \mathscr{SOS}(\mathscr{P}_1, \ldots, \mathscr{P}_n)$, and
(ii) $\exists j \mid 1 \leqslant j \leqslant n \wedge x_{\mathscr{P}_j} \notin \bar{M}$.

Verifying the above items is feasible in polynomial time. Thus, the complementary problem of $SS\text{-}SOS_n$ is in NP.

(2) Hardness. Now we prove that a reduction from the NP-complete problem SOS_n to the complementary problem of $SS\text{-}SOS_n$ is feasible in polynomial time. Consider an atom x such that, for each i ($1 \leqslant i \leqslant n$), $x \notin Var(\mathscr{P}_i)$. It is easy to see that $\mathscr{SOS}(\mathscr{P}_1, \ldots, \mathscr{P}_n) \neq \emptyset$ iff there exists $\bar{M} \in \mathscr{SOS}(\mathscr{P}_1, \ldots, \mathscr{P}_n)$ such that $\{x_{\mathscr{P}_1}, \ldots, x_{\mathscr{P}_n}\} \nsubseteq \bar{M}$. Moreover, in case $\{x_{\mathscr{P}_1}, \ldots, x_{\mathscr{P}_n}\} \nsubseteq \bar{M}$, it results that $\exists j \mid 1 \leqslant j \leqslant n \wedge x_{\mathscr{P}_j} \notin \bar{M}$. Thus, SOS_n is polynomially reducible to the complementary problem of $SS\text{-}SOS_n$. □

Theorem 6
The problem $IS\text{-}SOS_n$ is coNP-complete.

Proof
It suffices to prove that the complementary problem of $IS\text{-}SOS_n$ is NP-complete. Such a problem may be described as follows:

Instance: An SOLP collection $C = \{\mathscr{P}_1, \ldots, \mathscr{P}_n\}$ and an atom x.
Question: Does it hold that there exists $\bar{M} \in \mathscr{SOS}(\mathscr{P}_1, \ldots, \mathscr{P}_n)$ such that $\forall j$ ($1 \leqslant j \leqslant n$), $x_{\mathscr{P}_j} \notin \bar{M}$, i.e. $\{x_{\mathscr{P}_1}, \ldots, x_{\mathscr{P}_n}\} \nsubseteq \bar{M}$?

(1) Membership. We need to guess a candidate social interpretation $\bar{M}$ for C and, then, to verify that

(i) $\bar{M} \in \mathscr{SOS}(\mathscr{P}_1, \ldots, \mathscr{P}_n)$, and
(ii) $\{x_{\mathscr{P}_1}, \ldots, x_{\mathscr{P}_n}\} \nsubseteq \bar{M}$.

Verifying the above items is feasible in polynomial time. Thus, the complementary problem of $IS\text{-}SOS_n$ is in NP.

(2) Hardness. Now we prove that a reduction from the NP-complete problem SOS_n to the complementary problem of $IS\text{-}SOS_n$ is feasible in polynomial time. Consider an atom x such that, for each i ($1 \leqslant i \leqslant n$), $x \notin Var(\mathscr{P}_i)$. It is easy to see that $\mathscr{SOS}(\mathscr{P}_1, \dots, \mathscr{P}_n) \neq \emptyset$ iff there exists $\bar{M} \in \mathscr{SOS}(\mathscr{P}_1, \dots, \mathscr{P}_n)$ such that $\{x_{\mathscr{P}_1}, \dots, x_{\mathscr{P}_n}\} \nsubseteq \bar{M}$. Thus, SOS_n is polynomially reducible to the complementary problem of $IS\text{-}SOS_n$. □

Theorem 7
The problem $SC\text{-}SOS_n$ is NP-complete.

Proof
(1) Membership. We need to guess a candidate social interpretation $\bar{M}$ for C and, then, to verify that

(i) $\bar{M} \in \mathscr{SOS}(\mathscr{P}_1, \dots, \mathscr{P}_n)$, and
(ii) $\{x_{\mathscr{P}_1}, \dots, x_{\mathscr{P}_n}\} \subseteq \bar{M}$.

Verifying the above items is feasible in polynomial time. Thus, the problem $SC\text{-}SOS_n$ is in NP.

(2) Hardness. Now we prove that a reduction from the NP-complete problem SOS_n exists and it is feasible in polynomial time. Consider the SOLP collection $C' = \{\tau(\mathscr{P}_1), \dots, \tau(\mathscr{P}_n)\}$, where, for each i ($1 \leqslant i \leqslant n$), $\tau(\mathscr{P}_i)$ is an SOLP program obtained from $\mathscr{P}_i$ as follows:

$$\tau(\mathscr{P}_i) = \mathscr{P}_i \cup \{x \leftarrow\},$$

where it holds that, for each i ($1 \leqslant i \leqslant n$), $x \notin Var(\mathscr{P}_i)$. It is easy to see that $\mathscr{SOS}(\mathscr{P}_1, \dots, \mathscr{P}_n) \neq \emptyset$ iff for each $\bar{M} \in \mathscr{SOS}(\mathscr{P}_1, \dots, \mathscr{P}_n)$, there exists $\bar{M}' \in \mathscr{SOS}(\tau(\mathscr{P}_1), \dots, \tau(\mathscr{P}_n))$ such that $\bar{M}' = \bar{M} \cup \{x_{\mathscr{P}_1}, \dots, x_{\mathscr{P}_n}\}$. Thus, SOS_n is polynomially reducible to $SC\text{-}SOS_n$. □

Theorem 8
The problem $IC\text{-}SOS_n$ is NP-complete.

Proof
(1. Membership). We need to guess a candidate social interpretation $\bar{M}$ for C and, then, to verify that

(i) $\bar{M} \in \mathscr{SOS}(\mathscr{P}_1, \dots, \mathscr{P}_n)$, and
(ii) $\exists j \mid 1 \leqslant j \leqslant n \wedge x_{\mathscr{P}_j} \in \bar{M}$.

Verifying the above items is feasible in polynomial time. Thus, the problem $IC\text{-}SOS_n$ is in NP.

(2. Hardness). Now we prove that a reduction from the NP-complete problem SOS_n exists and it is feasible in polynomial time. consider the SOLP collection $C' = \{\tau(\mathscr{P}_1), \dots, \tau(\mathscr{P}_n)\}$, where, for each i ($1 \leqslant i \leqslant n$), $\tau(\mathscr{P}_i)$ is an SOLP program obtained from $\mathscr{P}_i$ as follows:

$$\tau(\mathscr{P}_i) = \mathscr{P}_i \cup \{x \leftarrow\},$$

where it holds that, for each i ($1 \leqslant i \leqslant n$), $x \notin Var(\mathscr{P}_i)$. It is easy to see that $\mathscr{SOS}(\mathscr{P}_1, \ldots, \mathscr{P}_n) \neq \emptyset$ iff for each $\bar{M} \in \mathscr{SOS}(\mathscr{P}_1, \ldots, \mathscr{P}_n)$, there exists $\bar{M}' \in \mathscr{SOS}(\tau(\mathscr{P}_1), \ldots, \tau(\mathscr{P}_n))$ such that $\bar{M}' = \bar{M} \cup \{x_{\mathscr{P}_1}, \ldots, x_{\mathscr{P}_n}\}$. Moreover, since $\{x_{\mathscr{P}_1}, \ldots, x_{\mathscr{P}_n}\} \subseteq \bar{M}'$, there exists j ($1 \leqslant j \leqslant n$) such that $x_{\mathscr{P}_j} \in \bar{M}'$. Thus, SOS_n is polynomially reducible to $IC\text{-}SOS_n$. □

7 Knowledge representation with SOLP programs

In this section, we provide interesting examples showing the capability of our language of representing common knowledge.

Example 14 (*Seating*)
We must arrange a seating for a number n of agents (representing, for instance, people invited to the wedding party introduced in Example 1), with m tables and a maximum of c chairs per table. Agents who like each other should sit at the same table; agents who dislike each other 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 Ith agent is represented by a predicate $agent(I)$ and his knowledge base is enclosed in a single SOLP program. Each program will include both a set of common rules encoding the problem and the agent's own requirements. The predicate $like(A)$ [resp. $dislike(A)$] means that the agent A is desired (resp. not tolerated) at the same table. $table(T)$ represents a table ($1 \leqslant T \leqslant m$) and $at(T)$ expresses the desire to sit at table T. For instance, the program $\mathscr{P}_1$ (which is associated with the agent 1) could be written as follows:

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

where the rules from r_1 to r_5 are common to all the programs (of course, the argument of the predicate $agent()$ in r_1 is suited to the enclosing program) and the rules r_6–r_9 express the agent's own requirements. In detail, the rule r_2 states that any agent cannot be seated at more than one table, the rule r_3 means that agent 1 sits at a particular table T if at least c agents he likes are seating at that table (c is a given constant). The rule r_4 states that it is forbidden that agent 1 shares a table with at least one or more agents he dislikes.

The rule r_5 provides consistency for the predicates *like* and *dislike*, while examples of such predicates are reported in rules r_6 and r_7. 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 the 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 15 (*Room arrangement*)
Consider a house having m rooms. We have to distribute some objects (i.e. furniture and appliances) over the rooms in such a way that we do not exceed the maximum number of objects, say c, allowed per room. Constraints about the color and/or the type of objects sharing the same room can be introduced. We assume that each object is represented by a single program encoding both the properties and the constraints we want to meet. Consider the following program:

$$\begin{array}{lrl}
r_1: & name(cupboard) \leftarrow & \\
r_2: & type(furniture) \leftarrow & \\
r_3: & color(yellow) \leftarrow & \\
r_4: & \leftarrow & at(R1), at(R2), R1 <> R2 \\
r_5: & at(R) \leftarrow & [,2]\{at(R), type(appliance), color(yellow), \\
 & & \quad [1,1]\{name(fridge)\}\}, room(R) \\
r_6: & at(R) \leftarrow & [,c-3]\{at(R), type(furniture)\}, room(R) \\
r_7: & \leftarrow & at(R), [1,]\{at(R), color(green)\},
\end{array}$$

where the properties of the current object are encoded as predicates representing the *name* (fridge, cupboard, table, etc.), the *type* (furniture or appliance), the *color*, and so on (see rules r_1–r_3). 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 two yellow appliances to share the room with the cupboard, provided 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 $c-3$, where c (representing the maximum number of objects per room) is given. Finally, the rule r_7 states that the cupboard cannot share the room with any green object.

Example 16 (*FPGA Design*)
In this example, we represent an extended version of a well-known problem belonging to the setting of FPGA (field programmable gate arrays) design, namely, *placement*. FPGAs are generic, programmable digital devices providing, in a single system, a way for digital designers to access thousands or millions of logic gates arranged in multilevel structures, referred to as *modules*, and to program them as desired by the end user. Placement consists in determining the module positions within the design area according to given constraints.

Consider a team of n electronic engineers, jointly working on a common FPGA design. Each of them is responsible for placing a given number of modules inside the chip design area, which is represented by a square grid of cells. In particular, each designer must meet a number of constraints concerning either (resp. both) his own modules or (resp. and) the modules of other designers. Moreover, the total chip area which is occupied by the modules must either match a given value or be less than a given value.

This setting can be encoded by a collection of SOLP programs[9] $C = \{Q_1, \ldots, Q_n, P\}$, where for each i ($1 \leqslant i \leqslant n$), Q_i describes designer i's requirements, and P

[9] Observe that, according to the implementation of the language that relies on DLV (Leone *et al.*, 2002); (Dell'Armi *et al.*, 2003), individual programs of our SOLP collection adopt the syntax of DLV, allowing both built-in predicates and standard aggregate functions.

represents an agent aimed to find admissible solutions to the placement problem. Such solutions are included into the social models of the SOLP collection C. In the following paragraphs we describe such programs in detail.

We assume that each module is rectangular-shaped and it is described by a predicate $mod(M, X, Y)$, where M is an identifier and X (resp. Y) is the horizontal (resp. vertical) module size measured in grid cells. First, we describe the program Q_i (corresponding to rules r_1–r_{28}), encoding designer i's placement constraints.

We distinguish among *hard* and *soft* constraints, respectively. Hard constraints are common to each program Q_i $(1 \leqslant i \leqslant n)$ and describe the placement problem. Soft constraints represent both requirements of a single designer on his own module's properties and requirements on the properties of modules owned by another designer.

Hard Constraints. The following rules encode the placement problem:

$$\begin{array}{llcl}
r_1: & grid(n,n) & \leftarrow & \\
r_2: & & \leftarrow & place(B,X,Y), unplaced(B,X,Y) \\
r_3: & place(B,X,Y) & \leftarrow & not\ unplaced(B,X,Y), mod(B,\_,\_), \#int(X), \\
 & & & \#int(Y), X < XG, Y < YG, grid(XG,YG) \\
r_4: & unplaced(B,X,Y) & \leftarrow & not\ place(B,X,Y), mod(B,\_,\_), \#int(X), \\
 & & & \#int(Y), X < XG, Y < YG, grid(XG,YG) \\
r_5: & & \leftarrow & place(B,X,Y), mod(B,XB,YB), grid(XG,YG), \\
 & & & XT > XG, +(X,XB,XT) \\
r_6: & & \leftarrow & place(B,X,Y), mod(B,XB,YB), grid(XG,YG), \\
 & & & YT > YG, +(Y,YB,YT) \\
r_7: & & \leftarrow & \#count\{B : place(B,\_,\_), mod(B,\_,\_), B = B1\} = 0, \\
 & & & mod(B1,\_,\_) \\
r_8: & & \leftarrow & place(B,X,Y), place(B,X1,Y1), X <> X1 \\
r_9: & & \leftarrow & place(B,X,Y), place(B,X1,Y1), Y <> Y1 \\
r_{10}: & & \leftarrow & cell(B,X,Y), cell(B1,X,Y), B <> B1 \\
r_{11}: & cell(B,X,Y) & \leftarrow & mod(B,XB,YB), place(B,X1,Y1), \#int(X), \\
 & & & \#int(Y), X1 \leqslant X, X < SX, +(X1,XB,SX), \\
 & & & Y1 \leqslant Y, Y < SY, +(Y1,YB,SY) \\
r_{12}: & & \leftarrow & cell(B,X,Y), [1,]\{cell(B1,X1,Y1)\}, X = X1, Y = Y1
\end{array}$$

First, the grid sizes are declared (rule r_1) and then, after candidate module positions are guessed (rules r_2–r_4), several requirements are checked: (i) a module cannot be placed outside the chip design area (rules $r_5 and r_6$); (ii) all modules must be placed (rule r_7); (iii) each module cannot be placed more than once (rules $r_8 and r_9$); (iv) modules owned by the same designer cannot overlap (rule r_{10}). By means of rule r_{11} the predicate $cell(B, X, Y)$ is true if module B covers the grid cell at coordinates X, Y. Finally, the intended meaning of the social rule r_{12} is to avoid overlapping of modules owned by different designers.

Soft Constraints. The following rules describe examples of constraints that designer i can specify on his own module's properties, i.e. setting either the absolute module position (rule r_{13}) or that relative to other modules (rules r_{14}–r_{17}). For instance, rules r_{14} and r_{15} specify that both modules 1 and 3 must be placed (a) on the same row (represented by the coordinate Y), and (b) such that module 1 is on the left of module 3. Finally, rules r_{16} and r_{17} require that module 1 is placed either 0 or 1 cell far from module 3. Note that the predicate $place(B, X, Y)$ sets the upper-left corner coordinates of module B to X, Y.

$$\begin{array}{lll}
r_{13}: & place(1,3,4) \leftarrow & \\
r_{14}: & \leftarrow & place(1,X,Y), place(3,X1,Y1), Y <> Y1 \\
r_{15}: & \leftarrow & place(1,X,Y), place(3,X1,Y1), X \geqslant X1 \\
r_{16}: & \leftarrow & place(1,X,Y), place(3,X1,Y1), \#int(X), \\
 & & mod(1,XB,YB), +(X,XM,X1), +(XB,D,XM), D < 0 \\
r_{17}: & \leftarrow & place(1,X,Y), place(3,X1,Y1), \#int(X), \\
 & & mod(1,XB,YB), +(X,XM,X1), +(XB,D,XM), D > 1
\end{array}$$

In addition, it is possible to encode, by means of social rules, the dependence of designer i's module properties from those of other designers. For instance, given an integer d, by means of the following rules designer i requires that module 4 is placed on the same row (rule r_{18}) as designer j's module 1 and such that a distance of exactly d cells exists between them (rules r_{19}, r_{20}).

$$\begin{array}{lll}
r_{18}: & \leftarrow & place(4,X4,Y4), [Q_j]\{place(1,X1,Y1)\}, Y4 <> Y1. \\
r_{19}: & \leftarrow & place(4,X4,Y4), [Q_j]\{place(1,X1,Y1)\}, D <> d, +(X4,XM,X1), \\
 & & X4 \leqslant X1, +(XB4,D,XM), mod(4,XB4,YB4) \\
r_{20}: & \leftarrow & place(4,X4,Y4), [Q_j]\{place(1,X1,Y1)\}, D <> d, +(X1,XM,X4), \\
 & & X1 \leqslant X4, +(XB1,D,XM), mod(1,XB1,YB1)
\end{array}$$

In order to ensure that all modules are properly spaced, rules r_{21}–r_{24} (resp. rules r_{25}–r_{28}) require that modules owned by designer i (resp. owned by designers i and j such that $j \neq i$) are mutually spaced by at least k cells, where k is a given integer constant.

$$\begin{array}{lll}
r_{21}: & \leftarrow & cell(B,X,Y), cell(B1,X1,Y1), B <> B1, \#int(D), +(X,D,X1), D < k \\
r_{22}: & \leftarrow & cell(B,X,Y), cell(B1,X1,Y1), B <> B1, \#int(D), +(X1,D,X), D < k \\
r_{23}: & \leftarrow & cell(B,X,Y), cell(B1,X1,Y1), B <> B1, \#int(D), +(Y,D,Y1), D < k \\
r_{24}: & \leftarrow & cell(B,X,Y), cell(B1,X1,Y1), B <> B1, \#int(D), +(Y1,D,Y), D < k \\
r_{25}: & \leftarrow & cell(B,X,Y), [1,]\{cell(B1,X1,Y1)\}, \#int(D), +(X,D,X1), D < k \\
r_{26}: & \leftarrow & cell(B,X,Y), [1,]\{cell(B1,X1,Y1)\}, \#int(D), +(X1,D,X), D < k \\
r_{27}: & \leftarrow & cell(B,X,Y), [1,]\{cell(B1,X1,Y1)\}, \#int(D), +(Y,D,Y1), D < k \\
r_{28}: & \leftarrow & cell(B,X,Y), [1,]\{cell(B1,X1,Y1)\}, \#int(D), +(Y1,D,Y), D < k
\end{array}$$

Now we describe the SOLP program P (rules r_{29}–r_{38}), representing an agent which collects from the designers admissible solutions to the placement problem. Moreover, by means of additional rules (r_{39} and r_{40}), P possibly requires that the placement layout area either is less than or matches a given value.

$$\begin{array}{lrl}
r_{29}: & pcell(X,Y) \leftarrow & [1,]\{cell(\_,X1,Y1)\}, X = X1, Y = Y1 \\
r_{30}: & exists\_x\_lt(W) \leftarrow & pcell(W,\_), pcell(W1,\_), W1 < W \\
r_{31}: & lowest\_x(X) \leftarrow & pcell(X,\_),\ not\ exists\_x\_lt(X) \\
r_{32}: & exists\_y\_lt(W) \leftarrow & pcell(\_,W), pcell(\_,W1), W1 < W \\
r_{33}: & lowest\_y(Y) \leftarrow & pcell(\_,Y),\ not\ exists\_y\_lt(Y) \\
r_{34}: & exists\_x\_ht(W) \leftarrow & pcell(W,\_), pcell(W1,\_), W1 > W \\
r_{35}: & highest\_x(X) \leftarrow & pcell(\_,X,\_),\ not\ exists\_x\_ht(X) \\
r_{36}: & exists\_y\_ht(W) \leftarrow & pcell(\_,W), pcell(\_,W1), W1 > W \\
r_{37}: & highest\_y(Y) \leftarrow & pcell(\_,Y),\ not\ exists\_y\_ht(Y) \\
r_{38}: & design\_area(A) \leftarrow & *(B,H,A), +(X1,B,X2), +(Y1,H,Y2), lowest\_x(X1), \\
 & & highest\_x(X2), lowest\_y(Y1), highest\_y(Y2)
\end{array}$$

Social rule r_{29} collects admissible solutions to the placement problem. The rules from r_{30} to r_{37} are used to represent the smallest rectangle enclosing all the placed modules. Then, the actual design area is computed by rule r_{38}.

In case an an upper bound b to be satisfied (resp. an exact value s to be matched) is given, then the following rule r_{39} (resp. r_{40}) may be added:

$$\begin{array}{rrcl} & r_{39}: & \leftarrow & design\_area(A), A > b \\ \text{(resp.} & r_{40}: & \leftarrow & design\_area(A), A <> s) \end{array}$$

Example 17 (*Contextual Reasoning*)
It is interesting to observe that SOLP programs can represent a form of contextual reasoning (Ghidini and Giunchiglia, 2001).

Although many definitions of the notion of *context* exist in the Artificial Intelligence literature (McCarthy, 1993; Brézillon, 1999; Ghidini and Giunchiglia, 2001), we can informally say that a context is an environment (i.e. a set of facts and the logic rules to perform inference with) in which the reasoning takes place.

In particular, in Ghidini and Giunchiglia (2001) two key principles of contextual reasoning are stated: *locality* (the reasoning task uses only a subset of the total knowledge available) and *compatibility* (additional constraints among different contexts may be specified to declare those which are mutually compatible).

Under this perspective, we are interested in representing this feature of common-sense reasoning, that is, given a problem to be solved, (i) bounding the reasoning to the knowledge which is strictly needed, the so-called *context* of the problem, and (ii) in case the original context is not suitable to reach a solution, enabling the use of new information provided by other contexts.

It is interesting to note that SOLP programs are well suited to represent contexts, since each of them enables reasoning which takes place both locally, i.e. at the level of the program knowledge base, and at the level of the other programs' knowledge bases. Thus, it is easy to model reasoning which involves several different contexts.

For instance, consider an SOLP program P_0 (representing the context of the agent A_0) including a social rule of the form

$$action \leftarrow [P_1]b_1, \ldots, [P_n]b_n,$$

meaning that the agent A_0 infers the term *action* if each term b_i ($1 \leqslant i \leqslant n$) is inferred by the corresponding agent A_i, i.e. b_i is part of an autonomous fixpoint of P_i. This way, the notion of locality is realized by representing each different context in a separate SOLP program, and the compatibility principle is pursued by suitably using SCs where member selection conditions identify contexts.

As an example, consider the well-known "Three Wise Men Puzzle," first introduced in Konolige (1984).

A king wishes to determine which of his three wise men is the wisest. He arranges them in a circle so that they can see and hear each other and tells them that he will put a white or a black spot on each of their foreheads but that at least one spot will be white. He then repeatedly asks them, "Do you know the colour of your spot?". What do they answer?

We represent by means of SOLP programs a slightly simpler version of the puzzle, where only two wise men are involved

r_1 : $color(white) \leftarrow$
r_2 : $color(black) \leftarrow$

King (agent K)

r_3 : $wise\_man(1..2) \leftarrow$
r_4 : $put\_spot(A, black) \leftarrow$ $not\ put\_spot(A, white), wise\_man(A)$
r_5 : $put\_spot(A, white) \leftarrow$ $not\ put\_spot(A, black), wise\_man(A)$
r_6 : $\leftarrow$ $\#count\{S : put\_spot(S, black)\} = N,$
$\#count\{A : wise\_man(A)\} = N, \#int(N)$
r_7 : $ask\_question(1) \leftarrow$
r_8 : $ask\_question(T2) \leftarrow$ $ask\_question(T1), \#succ(T1, T2), \#int(T1), \#int(T2)$
$not\ [1,]\{answer(white, T1), answer(black, T1)\},$

Wise man 1 (agent $W1$)

r_9 : $forehead(C) \leftarrow$ $[K]\{put\_spot(1, C)\}, color(C)$
r_{10} : $answer(white, 1) \leftarrow$ $[K]\{ask\_question(1)\}, [W2]\{forehead(black)\}$
r_{11} : $answer(black, 1) \leftarrow$ $[K]\{ask\_question(1)\}, [W2]\{forehead(white),$
$answer(white, 1)\}$
r_{12} : $answer(white, 2) \leftarrow$ $[K]\{ask\_question(2)\}, [W2]\{forehead(white),$
$not\ answer(white, 1), not\ answer(black, 1)\}$

Rules r_1 and r_2 are common to each SOLP program. Such rules set the admissible spot colors. Rules r_3–r_8 represent the king. Rule r_3 sets the number of wise men (two in this case). By means of rules r_4 and r_5, the king nondeterministically puts a spot on each wise man's forehead. Rule r_6 represents the king's statement "At least one spot is white." The king asks the question for the first time (rule r_7) and, after he has asked the question, if no agent gives an answer, then he asks the question again (rule r_8). Rule r_9 is used to store into the predicate *forehead* the information about the color of the corresponding wise man's spot. Observe that since each wise man cannot look at his own forehead, then the predicate *forehead* is further referenced in SCs only. Rules r_{10} and r_{11} represent the case of exactly one white spot: After the first time the king asks the question, if the first wise man sees a black spot on the other wise man's forehead, then he concludes that his spot is white (rule r_{10}). Otherwise, if the second wise man both has a white spot on his forehead and he answers "white," then the first wise man can conclude that the other wise man has seen a black spot on his forehead. Thus, the first wise man answers "black." Finally, rule r_{12} represents the case of two white spots. In such a case, after the first question, no wise man can conclude anything about the color of his own spot. After the second time the king asks the question, each wise man can answer "white" in case he sees a white spot and the other wise man has not answered the king's previous question. The correctness of such a statement can be proved by contradiction: If a wise man had a black spot on his forehead, then the other wise man would have seen it and, thus, he also would have answered "white" after the king's first question.

The SOLP program representing the second wise man (program $W2$) is easily obtained from $W1$ by exchanging the role of the two wise men, i.e. by replacing each occurrence of the program identifier $W2$ by $W1$.

For the sake of the simplicity we have considered a simple scenario, i.e. two wise men. It is possible to extend the reasoning encoded in the above programs, in order to write a general program for *n* wise men, by exploiting the nesting feature of the social conditions in such a way that reasoning on both the content and the temporal sequence of the wise men's statements is enabled.

8 Related work

Contextual Reasoning. As pointed out in Section 7, a relationship exists between our work and Ghidini and Giunchiglia (2001), where the *local model semantics* (LMS) is proposed to reason about contexts.

A survey covering the use of contexts in many fields of Artificial Intelligence can be found in Brézillon (1999). An approach concerning contextual reasoning and agent-based systems can be found in De Saeger and Shimojima (2006).

In McCarthy (1993), the author discusses the notion of context in Artificial Intelligence in order to solve the problem of *generality*, that is every logic theory is valid within the bounds of a definite context and it is possible to design a more general context where such a theory is not valid anymore.

Other approaches have been proposed in Buvač and Mason (1993) and Ghidini and Giunchiglia (2001). Moreover, these two works are compared in Serafini and Bouquet (2004). In the former work, the authors introduce the *propositional logic of context*, a modal logic aiming at formalizing McCarthy's ideas.

An approach which is more closely related to ours is proposed in Ghidini and Giunchiglia (2001), where the authors consider a set of logic languages, each representing a different context, and a suitable semantics is used to select among sets of *local* models, i.e. models pertaining to a single language, those which satisfy a given *compatibility* condition. Moreover, a proof-theoretical framework for contextual reasoning, called *multicontext systems*, is introduced where the notions of locality and compatibility are respectively captured by *inference* rules, whose scope is the single language, and *bridge* rules establishing relationships among different languages.

Observe that, in the previous section, we have shown that such a machinery can be represented by social rules where the body includes only member-selection-condition-based SCs, each corresponding to a different context to be included into the reasoning task. As a consequence, we argue that social rules are more general than bridge rules, since the former provide also (possibly nested) cardinal-selection-condition-based SCs. Since our work is not aimed to reason on contexts, a direct comparison with Ghidini and Giunchiglia (2001) cannot be done, although some correspondences may be found between the model-theoretical formalizations of both the local model semantics and the social semantics. In particular, we feel that the latter could be easily adapted to fully enable contextual reasoning inside logic programming. This is left for future work.

Logic-based multiagent systems. A related approach, where the semantics of a collection of abductive logic agents is given in terms of the stability of their interaction can be found in Bracciali *et al.* (2004) where the authors define the semantics of a multiagent system via a definition of *stability* on the set of all actions performed by all agents in the system, possibly arising from their communication and interaction via observation. According to the authors, a set of actions committed by different agents is *stable* if, assuming that an "oracle" could feed each of the agents with all the actions in the set performed by the other agents, each agent would do exactly what is in the set. We believe that such a machinery is similar to our fixpoint-based approach since we guess candidate social interpretations and select those which are compatible with the SCs of the SOLP collection.

In our work, we focused on the formalization of the semantics, assuming a perfect communication among the agents in such a way that each agent is able to know the mental state of the others. Using a different approach from ours, in Satoh and Yamamoto (2002), in order to face the possible incompleteness of information due to communication failures or delays in a multiagent system, a default hypothesis is used as a tentative answer and the computation continues until a reply is received which contradicts with the default.

Another interesting work is the MINERVA agent architecture (Leite *et al.*, 2002), based on dynamic logic programming (Alferes *et al.*, 2000). MINERVA is a modular architecture, where every agent is composed of specialized subagents that execute special tasks, e.g., reactivity, planning, scheduling, belief revision, and action execution. A common internal knowledge base, represented as one or more multidimensional dynamic logic programs (MDLP) (Alferes *et al.*, 2002), is concurrently manipulated by its specialized subagents. The MDLPs may encode object-level knowledge, or knowledge about goals, plans, intentions, etc.

The DALI project (Costantini and Tocchio, 2002) is a complete multiagent platform written entirely in Prolog. A DALI program results in an agent which is capable of reactive and proactive behavior, triggered by several kinds of events. The semantics of a DALI program is defined in terms of another program, where reactive and proactive rules are reinterpreted as standard Horn Clause rules.

Laima (De Vos *et al.*, 2005) agents are represented as ordered choice logic programs (OCLP) (De Vos, 2003) for modeling their knowledge and reasoning capabilities. Communication between the agents is regulated by unidirectional channels transporting information based on their answer sets.

IMPACT (Subrahmanian *et al.*, 2000) is an agent platform where programs may be used to specify what an agent is either obliged to do, may do, or cannot do on the basis of deontic operators of permission, obligation, and prohibition. IMPACT is grounded on a solid semantic framework based on the concept of feasible status set, which describes a set of actions dictated by an agent program that is consistent with the obligations and restrictions on the agent itself. Agent programs define integrity constraints, which must be satisfied in order to provide a feasible status set. The adoption of a logic-programming-based formalism, and the use of integrity constraints to define a feasible status set, guarantee agents to behave in a way that some desired properties hold.

Societies Of ComputeeS (SOCS) (Alberti *et al.*, 2004) is a project that was funded by European Union. The idea is to provide a computational logic model for the description, analysis, and verification of global and open societies of heterogeneous *computees*. The computee model is proposed as a full-fledged agent model, based on extended logic programming, allowing to define and study properties that can be enforced by its operational model.

Since our approach relies on the general notion of social behavior, it is of course interesting to illustrate how this concept is dealt with in the related field of intelligent agents, in order to make evident that—even in such context—this notion takes an important role. Indeed, beside *autonomy*, intelligent agents (Wooldridge and Jennings, 1995; Wooldridge, 2000) may be required to have *social ability*. The meaning of this concept is twofold: (1) the presence of a common language for communication, and (2) the capability of reasoning on the content of communication acts. Concerning item (1), KQML (Mayfield and Finin, 1995), and FIPA ACL (Cost *et al*., 2001), both based on the *speech act theory* (Cohen and Levesque, 1990), represent the main efforts done in the last years. The state-of-the-art literature on item (2) is represented by (Wooldridge, 2000; van der Hoek and Wooldrige, 2003; Mascardi *et al*., 2004). Social ability allows thus the agent individuals to have *beliefs*, *desires*, and *intentions* (BDI) (Rao and Georgeff, 1995; Rao, 1996) as a result of both the mutual communication and the consequent individual reasoning.

9 Conclusions and future work

In this paper, we have proposed a new language, *social logic programming* (SOLP), which extends compromise logic programming and enables social behavior among a community of individuals whose reasoning is represented by logic programs. A rich set of examples shows that the language has very nice capabilities of representing such a kind of knowledge. Moreover, we have given a translation from SOLP to logic programming with aggregates and discussed the computational complexity of several decision problems related to the social semantics.

Basically, the present paper gives the theoretical core for a multiagent oriented software environment, including suitable specialized features, like information hiding, speech-act mechanisms, security, and so on. However, these issues are interesting directions of our future work.

For instance, information hiding can be implemented as follows. Given an SOLP collection C, we make the following assumption: by default each agent cannot see into other agents' mind, that is all atoms in each SOLP program are viewed only by the program itself, i.e. they are private. In order to make some atom public we could add a suffix, say P, to such an atom, i.e. a is meant as private, while aP is meant as the public version of a. Then, by means of a suitable modification of our translation machinery, any social condition can be activated only on public atoms.

An agent communication machinery can be conceived that relies on the above feature. In case an agent wants to send a message to another agent, then the former could make public a suitable set of atoms in such a way that they are visible only to the latter. This approach could be easily extended to the scenario where one agent wants to send a message to either a group of agents or to the whole community.

Another feature we intend to include into future extensions of SOLP is the representation of evolving agent mental states. We believe that a collection of SOLP programs could be easily managed by some existing logic framework which is tailored to program update or belief revision tasks, such as, for instance, dynamic logic programming (DLP) (Alferes *et al.*, 2000). The resulting system should work in a cyclic fashion: (i) social models of the SOLP collection are computed, (ii) by exploiting DLP, the SOLP programs within the collection are possibly updated according to the intended evolution of agent beliefs and intentions, (iii) the cycle restarts. Starting from the basic approach proposed in Section 7 (Example 16), another interesting issue to investigate is the capability of the language, in the general case, of representing cooperative approaches to solving combinatorial optimization problems, possibly by introducing some suitable extensions.

Finally, we plan to enhance the language of SOLP by adding both classical negation and rule-head disjunction. We expect the former to be easy to implement, while the integration of latter requires some preliminary study, as the introduction of disjunction in logic programming always results in a growth of the language expressivity toward higher levels in the computational complexity hierarchy.

Appendix A

In order to improve the overall readability of the paper, a number of lemma and theorem proofs are described in this section.

Lemma 1 (see page 660) Given an SOLP collection $SP = \{\mathscr{P}_1,\ldots,\mathscr{P}_n\}$, an integer j $(1 \leqslant j \leqslant n)$, an SOLP program $\mathscr{P}_j \in SP$, a social interpretation $\bar{I}$ for SP, and an SC $s \in MSC^{\mathscr{P}_j}$, it holds that s is true for $\mathscr{P}_j$ w.r.t. $\bar{I}$ iff $\exists M \in SM(C(\mathscr{P}_1,\ldots,\mathscr{P}_n) \cup Q)$ s.t. $\rho(s)_{\mathscr{P}_j} \in M$, where $Q = \{a \leftarrow\mid a \in \bar{I}\}$.

Proof

Before starting with the proof, let us denote the set of all the SCs occurring in s (plus the SC s itself) by $N(s)$, and the LP$^{\mathscr{A}}$ program $C(\mathscr{P}_1,\ldots,\mathscr{P}_n)$ by $\bar{C}$.

($\Rightarrow$). We have to prove that if s is true for $\mathscr{P}_j$ w.r.t. $\bar{I}$, then there exists a stable model M of the logic program $\bar{C} \cup Q$ s.t. the atom $\rho(s)_{\mathscr{P}_j}$, corresponding to s by means of the translation, is included in M. We proceed by induction on the maximum nesting depth (see p. 11) of the SCs in $N(s)$, $d = max_{s' \in N(s)}\{depth(s')\}$ $(d \geqslant 0)$.

(*Basis*). In case $d = 0$, then $N(s) = \{s\}$ and $depth(s) = 0$ (recall the definition of the function depth on p. 11). Since $|N(s)| = 1$, s is a simple SC, i.e. $skel(s) = \emptyset$. Now, assume by contradiction that s is true for $\mathscr{P}_j$ w.r.t. $\bar{I}$ and that for each $M \in SM(\bar{C} \cup Q)$, $\rho(s)_{\mathscr{P}_j} \notin M$. Observe now that it may occur either in the following cases: (1) $cond(s) = [\mathscr{P}_k](1 \leqslant k \leqslant n)$, or (2) $cond(s) = [l,h]$.

In case (1), according to Definition 13, there exists a set $S = \{r_1, r_2\} \subseteq \bar{C} \cup Q$ such that

$$\begin{array}{rrcl} r_1: & \rho(s)_{\mathscr{P}_j} & \leftarrow & (g(s))(k)_{\mathscr{P}_j} \\ r_2: & (g(s))(k)_{\mathscr{P}_j} & \leftarrow & \bigwedge_{b \in content(s)} b_{\mathscr{P}_k} \end{array}$$

and for each $r \in (\bar{C} \cup Q) \setminus S$, $head(r) \neq r_1 \wedge head(r) \neq r_2$.

Since we have assumed that $\rho(s)_{\mathscr{P}_j} \notin M$, it is easy to see that both $body(r_1)$ and $body(r_2)$ are false w.r.t. M. Moreover, it holds that for each $M \in SM(\bar{C} \cup Q)$, $\bar{I} \subseteq M$, as $Q = \{a \leftarrow\mid a \in \bar{I}\}$. As a consequence, $body(r_2)$ is false w.r.t. $\bar{I}$.

Since the elements occurring in $body(r_2)$ are labeled literals, we have proven that $cond(s) = [\mathscr{P}_k]$ and that the condition $\forall a \in content(s)$, a is true for $\mathscr{P}_k$ w.r.t. $\bar{I}$ does not hold. Such a result, according to Definition 8, contradicts the hypothesis that s is true for $\mathscr{P}_j$ w.r.t. $\bar{I}$ and, therefore, concludes the proof of the basis of the induction, in case (1).

In case (2), since $skel(s) = \emptyset$, according to Definition 13, there exists a set of rules $S = \{r_1\} \cup \{s_i \mid 1 \leqslant i \neq j \leqslant n\} \subseteq (\bar{C} \cup Q)$ such that

$$\begin{array}{llll} r_1 : & \rho(s)_{\mathscr{P}_j} \leftarrow & l \leqslant \#\texttt{count}\{K : (g(s))(K)_{\mathscr{P}_j}, K \neq j\} \leqslant h & \\ s_i : & (g(s))(i)_{\mathscr{P}_j} \leftarrow & \bigwedge_{b \in content(s)} b_{\mathscr{P}_i} & (1 \leqslant i \neq j \leqslant n). \end{array}$$

Moreover, for each $r \in (\bar{C} \cup Q) \setminus S$ and for each $t \in S$, $head(r) \neq head(t)$.

Now, since $\rho(s)_{\mathscr{P}_j} \notin M$, $body(r_1)$ is false w.r.t. M. Since $body(r_1) = l \leqslant \#\texttt{count}\{K : (g(s))(K)_{\mathscr{P}_j}, K \neq j\} \leqslant h$, according to the definition of aggregate functions Dell'Armi *et al.* (2003), for each $D \subseteq \{i \mid 1 \leqslant i \neq j \leqslant n\}$ $(l \leqslant |D| \leqslant h)$, there exists k s.t. $1 \leqslant k \neq j \leqslant n$ and $(g(s))(k)_{\mathscr{P}_j}$ is false w.r.t. M. Thus, there exists a rule $s_k \in \bar{C} \cup Q$ s.t. $s_k : (g(s))(k)_{\mathscr{P}_j} \leftarrow \bigwedge_{b \in content(s)} b_{\mathscr{P}_k}$ and $head(s_k)$ is false w.r.t. M. Since $\forall r \in (\bar{C} \cup Q) \setminus \{s_k\}$, $head(r) \neq head((s_k)$, $body(s_k)$ is false w.r.t. M, i.e. $\bigwedge_{b \in content(s)} b_{\mathscr{P}_k}$ is false w.r.t. M. Now, since $\bar{I} \subseteq M$, $\bigwedge_{b \in content(s)} b_{\mathscr{P}_k}$ is false w.r.t. $\bar{I}$.

Thus, we have proven that for each set $D \subseteq \{i \mid 1 \leqslant i \neq j \leqslant n\}$ s.t. $l \leqslant |D| \leqslant h$, there exists some $k \in D$ and some $x \in content(s)$ s.t. x is false for $\mathscr{P}_k$ w.r.t. $\bar{I}$. This result, according to item (2) of Definition 8, contradicts the hypothesis that s is true for $\mathscr{P}_k$ w.r.t. $\bar{I}$.

(*Induction*). Assume that the statement holds for $max_{s' \in N(s)}\{depth(s')\} = d > 0$ and consider the case $max_{s' \in N(s)}\{depth(s')\} = d + 1$. First, observe that s is not simple, because $skel(s) \neq \emptyset$. Since s is well formed, $cond(s) = [l, h]$. Thus, according to Definition 13, there exists a set of rules $S = \{r_1\} \cup \{s_i \mid 1 \leqslant i \neq j \leqslant n\} \subseteq (\bar{C} \cup Q)$ s.t.

$$\begin{array}{llll} r_1 : & \rho(s)_{\mathscr{P}_j} \leftarrow & l \leqslant \#\texttt{count}\{K : (g(s))(K)_{\mathscr{P}_j}, K \neq j\} \leqslant h & \\ s_i : & (g(s))(i)_{\mathscr{P}_j} \leftarrow & \bigwedge_{b \in content(s)} b_{\mathscr{P}_i} \wedge \bigwedge_{s' \in skel(s)} \rho(s')_{\mathscr{P}_j} & (1 \leqslant i \neq j \leqslant n). \end{array}$$

Now, observe that

(1) For each $s' \in skel(s)$, $max_{\sigma \in N(s')}\{depth(\sigma)\} = d$, and
(2) Since s is true for P_k w.r.t. $\bar{I}$, according to Definition 8, for each $s' \in skel(s)$, s' is true for $\mathscr{P}_j$ w.r.t. $\bar{I}$.

On the basis of the above observations and the induction hypothesis, it holds that for each $s' \in skel(s)$, $\rho(s')_{\mathscr{P}_j} \in M$.

Since s is true for $\mathscr{P}_j$ w.r.t. $\bar{I}$ and $cond(s) = [l, h]$, according to Definition 8 there exists $D \subseteq SP \setminus \{\mathscr{P}_j\}$ s.t. $l \leqslant |D| \leqslant h$ and $\forall a \in content(s)$, $\forall \mathscr{P} \in D$, a is true for $\mathscr{P}$ w.r.t. $\bar{I}$.

Thus, it holds that, for each $1 \leqslant i \leqslant n$ such that both $i \neq j$ and $\mathscr{P}_i \in D$, $\bigwedge_{b \in content(s)} b_{\mathscr{P}_i} \wedge \bigwedge_{s' \in skel(s)} \rho(s')_{\mathscr{P}_j}$ is true w.r.t. M, as $\bar{I} \subseteq M$. Since M is a stable model of $\bar{C} \cup Q$, according to the definition of the set S, for each i s.t. $1 \leqslant i \neq j \leqslant n$ and s.t. $\mathscr{P}_i \in D$, $(g(s))(i)_{\mathscr{P}_j}$ is true w.r.t. M.

Since $l \leqslant |D| \leqslant h$, there exists a set of literals $D' = \{(g(s))(i)_{\mathscr{P}_j} \mid 1 \leqslant i \neq j \leqslant n\}$ s.t. $l \leqslant |D'| \leqslant h$ and s.t. for each element $d \in D'$, d is true w.r.t. M.

Now, according to the definition of aggregate functions (Dell'Armi *et al.* 2003), $body(r_1)$ is true w.r.t. M and, since M is a stable model of $\bar{C} \cup Q$, $head(r_1)$ is true w.r.t. M, i.e. $\rho(s)_{\mathscr{P}_j}$ is true w.r.t. M. Such a result concludes the *only-if* part ($\Rightarrow$) of the proof.

($\Leftarrow$). We have to prove that if $\exists M \in SM(\bar{C} \cup Q) \mid \rho(s)_{\mathscr{P}_j} \in M$, then s is true for $\mathscr{P}_j$ w.r.t. $\bar{I}$. We proceed by induction on the maximum nesting depth of the elements in $N(s)$, $d = max_{s' \in N(s)}\{depth(s')\}$ ($d \geqslant 0$).

(*Basis*). In case $d = 0$, then $N(s) = \{s\}$ and $depth(s) = 0$. Since $|N(s)| = 1$, s is a simple SC, i.e. $skel(s) = \emptyset$. Observe now that it may occur either in the following cases: (1) $cond(s) = [\mathscr{P}_k](1 \leqslant k \leqslant n)$, or (2) $cond(s) = [l, h]$.

In case (1), according to Definition 13, there exist two rules in $\bar{C} \cup Q$ of the form

$$\begin{array}{lrcl} r_1 : & \rho(s)_{\mathscr{P}_j} & \leftarrow & (g(s))(k)_{\mathscr{P}_j} \\ r_2 : & (g(s))(k)_{\mathscr{P}_j} & \leftarrow & \bigwedge_{b \in content(s)} b_{\mathscr{P}_k} \end{array}$$

and such that $\forall r \in (\bar{C} \cup Q) \setminus \{r_1, r_2\}$, $head(r) \neq head(r_1) \wedge head(r) \neq head(r_2)$.

As a consequence, since $\rho(s)_{\mathscr{P}_j} \in M$, $body(r_1)$ is true w.r.t. M. Now, since M is a stable model of $\bar{C} \cup Q$, $body(r_2)$ is true w.r.t. M.

Now, observe that according to Definitions 12 and 13, for each $b \in content(s)$, $b_{\mathscr{P}_k} \notin (M \setminus \bar{I})$, because $M \setminus \bar{I}$ includes only literals that are auxiliary to the translation. As a consequence, $body(r_2)$ is true w.r.t. $\bar{I}$, i.e. for each $b \in content(s)$, b is true for $\mathscr{P}_k$ w.r.t. $\bar{I}$.

Now, we have proven that $cond(s) = [\mathscr{P}_k](1 \leqslant k \leqslant n)$ and there exists $\mathscr{P}_k \in SP$ s.t. for each $a \in content(s)$, a is true for $\mathscr{P}_k$ w.r.t. $\bar{I}$, thus, s is true for $\mathscr{P}_k$ w.r.t. $\bar{I}$. This concludes the proof of the basis of the induction, in case (1).

In case (2), since $skel(s) = \emptyset$, according to Definition 13, there exists a set of rules $S = \{r_1\} \cup \{s_i \mid 1 \leqslant i \neq j \leqslant n\} \subseteq (\bar{C} \cup Q)$ such that

$$\begin{array}{lrcll} r_1 : & \rho(s)_{\mathscr{P}_j} & \leftarrow & l \leqslant \#\mathtt{count}\{K : (g(s))(K)_{\mathscr{P}_j}, K \neq j\} \leqslant h & \\ s_i : & (g(s))(i)_{\mathscr{P}_j} & \leftarrow & \bigwedge_{b \in content(s)} b_{\mathscr{P}_i} & (1 \leqslant i \neq j \leqslant n). \end{array}$$

Moreover, for each $r \in (\bar{C} \cup Q) \setminus S$ and for each $t \in S$, $head(r) \neq head(t)$.

Since M is a stable model of $\bar{C} \cup Q$ and $\rho(s)_{\mathscr{P}_j} \in M$, $body(r_1)$ is true w.r.t. M. According to the definition of aggregate functions Dell'Armi *et al.* (2003), there exists a set of integers D s.t. $l \leqslant |D| \leqslant h$, for each i, $i \neq j$ and $\mathscr{P}_i \in SP$ and, finally, for each $i \in D$, $(g(s))(i)_{\mathscr{P}_j}$ is true w.r.t. M.

As a consequence, there exists a set of rules $D' = \{s_i \mid i \in D \wedge head(s_i) = (g(s))(i)_{\mathscr{P}_j}\} \subseteq S \setminus \{r_1\}$ s.t. for each rule $s_i \in D'$, $body(s_i)$ is true w.r.t. M. Since $\bar{I} \subseteq M$, it holds that for each $s_i \in D'$, $body(s_i)$ is true w.r.t. $\bar{I}$. Now, observe that $l \leqslant |D'| = |D| \leqslant h$ and that, according to Definition 13, for each $s_i \in D$, $body(s_i) = \bigwedge_{b \in content(s)} b_{\mathscr{P}_i}$. We have obtained that there exists a set D s.t. $l \leqslant |D| \leqslant h$, for each $i \in D$, $i \neq j$ and $\mathscr{P}_i \in SP$. Finally, it holds that for each $b \in content(s)$ and for each $i \in D$, b is true for $\mathscr{P}_i$ w.r.t. $\bar{I}$. Now, it results that there exists a set $\Delta = \{\mathscr{P}_i \mid i \in D \wedge \mathscr{P}_i \in SP\}$ s.t. $\Delta \subseteq SP \setminus \{\mathscr{P}_j\}$, $l \leqslant |Delta| = |D| \leqslant h$ and for each $b \in content(s)$, and for each $\mathscr{P} \in \Delta$, b is true for $\mathscr{P}_i$ w.r.t. $\bar{I}$.

According to Definition 8 we have proven that s is true for $\mathscr{P}_j$ w.r.t. $\bar{I}$. Such a result concludes the proof of the basis of the induction.

(*Induction*). Assume that the statement holds for $max_{s' \in N(s)}\{depth(s')\} = d > 0$. and consider the case $max_{s' \in N(s)}\{depth(s')\} = d+1$. First, observe that s is not simple. Since s is well formed, $cond(s) = [l, h]$. Thus, according to Definition 13, there exists a set of rules $S = \{r_1\} \cup \{s_i \mid 1 \leqslant i \neq j \leqslant n\} \subseteq (\bar{C} \cup Q)$ s.t.

$$\begin{array}{lrll} r_1 : & \rho(s)_{\mathscr{P}_j} \leftarrow & l \leqslant \#\mathtt{count}\{K : (g(s))(K)_{\mathscr{P}_j}, K \neq j\} \leqslant h & \\ s_i : & (g(s))(i)_{\mathscr{P}_j} \leftarrow & \bigwedge_{b \in content(s)} b_{\mathscr{P}_i} \wedge \bigwedge_{s' \in skel(s)} \rho(s')_{\mathscr{P}_j} & (1 \leqslant i \neq j \leqslant n). \end{array}$$

Moreover, for each $r \in (\bar{C} \cup Q) \setminus S$ and for each $t \in S$, $head(r) \neq head(t)$.

Now, observe that, since $\rho(s)_{\mathscr{P}_j} \in M$ and M is a stable model of $\bar{C} \cup Q$, according to the definition of aggregate functions Dell'Armi *et al.* (2003), there exists a set $\Delta' \subseteq \{(g(s))(i)_{\mathscr{P}_j} \mid 1 \leqslant i \neq j \leqslant n\}$ s.t. $l \leqslant |\Delta'| \leqslant h$ and $\forall x \in \Delta'$, x is true w.r.t. M. Thus, there exists $\Delta'' \subseteq S \setminus \{r_1\}$ s.t. $l \leqslant |\Delta''| = |\Delta'| = |\Delta| \leqslant h$ and s.t. for each $s_i \in \Delta''$, $head(s_i)$ is true w.r.t. M. Since M is a stable model of $\bar{C} \cup Q$, for each $s_i \in \Delta''$, $body(s_i)$ is true w.r.t. M. Now, note that for each i s.t. $s_i \in \Delta''$, $body(s_i) = \bigwedge_{b \in content(s)} b_{\mathscr{P}_i} \wedge \bigwedge_{s' \in skel(s)} \rho(s')_{\mathscr{P}_j}$.

Thus, for each i s.t. $s_i \in \Delta''$, $\bigwedge_{b \in content(s)} b_{\mathscr{P}_i}$ is true w.r.t. M and $\bigwedge_{s' \in skel(s)} \rho(s')_{\mathscr{P}_j}$ is true w.r.t. M.

Now, since $\bar{I} \subseteq M$ and according to Definition 13, for each i s.t. $s_i \in \Delta''$, $\bigwedge_{b \in content(s)} b_{\mathscr{P}_i}$ is true w.r.t. $\bar{I}$. Since for each i s.t. $s_i \in \Delta''$, $b_{\mathscr{P}_i}$ is a literal labeled w.r.t. $\mathscr{P}_i$, it is easy to see that for each i s.t. $s_i \in \Delta''$ and for each $b \in content(s)$, b is true for $\mathscr{P}_i$ w.r.t. $\bar{I}$. Moreover, by induction hypothesis, for each $s' \in skel(s)$, s' is true for $\mathscr{P}_j$ w.r.t. $\bar{I}$.

Thus, we have obtained that $cond(s) = [l, h]$ and there exists a set $D \subseteq \{i \mid 1 \leqslant i \neq j \leqslant n\}$ s.t. $l \leqslant |D| \leqslant h$ and for each $i \in D$ s.t. $\mathscr{P}_i \in SP$ and for each $a \in content(s)$, it holds that a is true w.r.t. $\mathscr{P}$ and for each $s' \in skel(s)$, s' is true for $\mathscr{P}_j$ w.r.t. $\bar{I}$. According to Definition 8, we have proven that s is true for $\mathscr{P}_j$ w.r.t. $\bar{I}$. This concludes the proof of the lemma. □

Theorem 2 (see page 664)
Given an SOLP collection $SP = \{\mathscr{P}_1, \ldots, \mathscr{P}_n\}$, it holds that $A = B$, where

$$\begin{array}{rll} A = & SM(P'_u \cup C(\mathscr{P}_1, \ldots, \mathscr{P}_n)) & \text{and} \\ B = & \{\bar{F} \cup \bar{G} \cup \bar{H} \mid & \\ (1) & \bar{F} = \bigcup_{1 \leqslant i \leqslant n} (F^i)_{\mathscr{P}_i} \wedge F^i \in AFP(\mathscr{P}_i) \wedge \bar{F} \in \mathscr{SOS}(\mathscr{P}_1, \ldots, \mathscr{P}_n) \wedge & \\ (2) & \bar{G} = \bigcup_{1 \leqslant i \leqslant n} (G^i)_{\mathscr{P}_i} \wedge (G^i)_{\mathscr{P}_i} = [F^i]_{\mathscr{P}_i} \setminus (F^i)_{\mathscr{P}_i} \wedge & \\ (3) & \bar{H} = \bigcup_{1 \leqslant i \leqslant n} (H^i)_{\mathscr{P}_i} \wedge (H^i)_{\mathscr{P}_i} = \bigcup_{s \in MSC^{\mathscr{P}_i}} SAT_{\bar{F}}^{\mathscr{P}_i}(s)\}. & \end{array}$$

Proof
Before starting with the proof, let us denote $C(\mathscr{P}_1, \ldots, \mathscr{P}_n)$ by $\bar{C}$.

($\subseteq$). By contradiction, assume that $\forall X \in A, X \notin B$. Observe that, according to Definitions 14, 17, and 20, it holds that $X = \bar{F} \cup \bar{G} \cup \bar{H}$ such that $\bar{F} \cap \bar{G} = \emptyset$, $\bar{G} \cap \bar{H} = \emptyset$, $\bar{F} \cap \bar{H} = \emptyset$. Thus, we prove that either condition (1), (2), or (3) is false. We consider each condition separately.

Condition (1). In case condition (1) is false, it holds that $\bar{F} \notin \mathscr{SOS}(\mathscr{P}_1, \ldots, \mathscr{P}_n)$. It follows that either

(a) $\exists i \mid (F^i)_{\mathscr{P}_i} \subseteq \bar{F} \wedge F^i \notin AFP(\mathscr{P}_i)$, or
(b) $\exists i, r \mid (F^i)_{\mathscr{P}_i} \subseteq \bar{F} \wedge F^i \in AFP(\mathscr{P}_i) \wedge r \in \mathscr{P}_i \wedge ST_C(\bar{F}) \neq \bar{F}$.

In case (a), by virtue of Lemma 5, it holds that $(F^i)_{\mathscr{P}_i} \notin SM(P_u)$. As a consequence, it is easy to see that $(F^i)_{\mathscr{P}_i} \notin SM(P'_u \cup \bar{C})$. Since $(F^i)_{\mathscr{P}_i} \in X$, then we have thus reached a contradiction.

If item (b) occurs, it holds that either

(b_1) $\exists a, \mathscr{P}_i \mid \mathscr{P}_i \in C \wedge a_{\mathscr{P}_i} \in \bar{F} \wedge a_{\mathscr{P}_i} \notin ST_C(\bar{F})$, or
(b_2) $\exists a, \mathscr{P}_i \mid \mathscr{P}_i \in C \wedge a_{\mathscr{P}_i} \notin \bar{F} \wedge a_{\mathscr{P}_i} \in ST_C(\bar{F})$.

In case (b_1), the following conditions are true

(i) $\forall r \in \mathscr{P}_i, body(r)$ is true w.r.t. $\bar{F} \Rightarrow head(r) \neq a$, and
(ii) $\forall r \in \mathscr{P}_i, body(r)$ is true w.r.t. $\bar{F} \wedge a$ is true for $\mathscr{P}_j$ w.r.t. $\bar{F} \Rightarrow head(r) \neq okay(a)$.

In case (i) of item (b_1), it holds that, according to Definitions 15 and 16, for each $r' \in S'_2(\hat{\mathscr{P}}_i)$, s.t. $body(r')$ is true w.r.t. X, it results that $head(r') \neq sa_{\mathscr{P}_i}$. Thus, $sa_{\mathscr{P}_i} \notin X$. As a consequence, according to the definition of $S'_3(\hat{\mathscr{P}}_i)$ (see Definition 16), it holds that $a_{\mathscr{P}_i} \notin X$. Since the hypothesis requires that $a_{\mathscr{P}_i} \in \bar{F}$ and $\bar{F} \subseteq X$, we have reached a contradiction.

Consider now case (ii) of item (b_1). It holds that, according to Definitions 15 and 16, for each $r' \in S'_2(\hat{\mathscr{P}}_i)$ s.t. $a_{\mathscr{P}_i} \wedge body(r')$ is true w.r.t. X, it results that $head(r') \neq sa_{\mathscr{P}_i}$. Thus, $sa_{\mathscr{P}_i} \notin X$. According to the definition of $S'_3(\hat{\mathscr{P}}_i)$, it holds that $a_{\mathscr{P}_i} \notin X$. Since the hypothesis requires that $a_{\mathscr{P}_i} \in \bar{F}$ and $\bar{F} \subseteq X$, we have reached a contradiction. This concludes the part of the proof concerning item (b_1) above.

Consider now item (b_2). In this case at least one of the following conditions holds:

(i) $\exists r \in \mathscr{P}_i \mid body(r)$ is true w.r.t. $\bar{F} \wedge a$ is true for $\mathscr{P}_j$ w.r.t. $\bar{F} \wedge head(r) = okay(a)$,
(ii) $\exists r \in \mathscr{P}_i \mid body(r)$ is true w.r.t. $\bar{F} \wedge head(r) = a$.

If case (i) of item (b_2) occurs, it holds that $a_{\mathscr{P}_i} \in \bar{F}$. Now, the contradiction is thus reached, since according to the hypothesis, $a_{\mathscr{P}_i} \notin \bar{F}$.

Consider now case (ii) of item (b_2). Let r be of the form $a \leftarrow b_1, \ldots b_v, s_1, \ldots, s_m$ (we do not lose in generality because r is any social rule). According to Definitions 15 and 16, there exists a rule $r' \in S'_2(\hat{\mathscr{P}}_i)$ such that r' has the form $sa_{\mathscr{P}_i} \leftarrow b^1_{\mathscr{P}_i}, \ldots, b^v_{\mathscr{P}_i}, \rho(s_1)_{\mathscr{P}_i}, \ldots \rho(s_m)_{\mathscr{P}_i}$.

Now, since $body(r)$ is true w.r.t. $\bar{F}$ and on the basis of results of Lemma 1, it holds that $body(r')$ is true w.r.t. X and $head(r') = sa_{\mathscr{P}_i}$. According to the definition of $S'_3(\hat{\mathscr{P}}_i)$ (see Definition 16), since $a_{\mathscr{P}_i} \notin X$, then it holds that $sa_{\mathscr{P}_i} \notin X$. Thus, $body(r')$ is true w.r.t. X and $head(r')$ is false w.r.t. X. As a consequence, there exists a rule r' in $P'_u \cup \bar{C}$ such that r' is false w.r.t. the model X $[X \in SM(P'_u \cup \bar{C})]$ which is a contradiction. The proof of condition (1) is thus concluded. Now let us prove condition (2).

Condition (2). In case condition (2) is false, there exists i s.t. $(G^i)_{\mathscr{P}_i} \neq [F^i]_{\mathscr{P}_i} \setminus (F^i)_{\mathscr{P}_i}$. Thus, $(F^i)_{\mathscr{P}_i} \cup (G^i)_{\mathscr{P}_i} \neq [F^i]_{\mathscr{P}_i}$ and then, according to Definition 20, $(F^i)_{\mathscr{P}_i} \cup (G^i)_{\mathscr{P}_i} \notin \bigcup_{F \in AFP(\mathscr{P}_i)} \{[F]_{\mathscr{P}_i}\}$. Now, by virtue of Lemmas 3 and 5, $(F^i)_{\mathscr{P}_i} \cup (G^i)_{\mathscr{P}_i} \notin SM(P_u)$ (recall that $P_u = \bigcup_{1 \leqslant i \leqslant n} \Gamma'(A(\hat{\mathscr{P}}_i)))$. According to Definitions 14, 16, and 17, it is easy to see that $(F^i)_{\mathscr{P}_i} \cup (G^i)_{\mathscr{P}_i} \notin SM(P'_u \cup \bar{C})$. Since $(F^i)_{\mathscr{P}_i} \cup (G^i)_{\mathscr{P}_i} \in X$, we have reached a contradiction. Consider now the last case.

Condition (3). If condition (3) is false, then there exists i such that $H^i_{\mathscr{P}_i}$ is not equal to $\bigcup_{s \in MSC^{\mathscr{P}_i}} SAT^{\mathscr{P}_i}_{\bar{F}}(s)$. Thus, it holds that either

(a) $\exists h \in (H^i)_{\mathscr{P}_i} \mid h \notin \bigcup_{s \in MSC^{\mathscr{P}_i}} SAT^{\mathscr{P}_i}_{\bar{F}}(s)$, or
(b) $\exists h \in \bigcup_{s \in MSC^{\mathscr{P}_i}} SAT^{\mathscr{P}_i}_{\bar{F}}(s) \mid h \notin (H^i)_{\mathscr{P}_i}$.

In case (a), according to Definition 21, since $h \notin \bigcup_{s \in MSC^{\mathscr{P}_i}} SAT^{\mathscr{P}_i}_{\bar{F}}(s)$, it holds that $\forall M \in SM(\bar{C} \cup Q)$ s.t. $Q = \{a \leftarrow\mid a \in \bar{F}\}$, it results that $h \notin M$. As a consequence, according to Definition 17, we have that $\forall M \in SM(P'_u \cup \bar{C})$, $h \notin M$. Now, we have reached a contradiction.

In case (b), according to Definition 21, h is either an auxiliary ρ-atom or a g-predicate and it holds that $h \notin (H^i)_{\mathscr{P}_i}$. Since $\bar{H} = \bigcup_{1 \leqslant k \leqslant n} (H^k)_{\mathscr{P}_k}$, and h is labeled w.r.t. $\mathscr{P}_i$, $h \notin \bar{H}$.

Now we have reached a contradiction, because $h \in X \setminus \bar{H} = \bar{F} \cup \bar{G}$ and $\bar{F} \cup \bar{G}$ does not include, according to conditions (1) and (2) of the theorem statement, either ρ-atoms or g-predicates.

($\supseteq$). By contradiction, assume that $\forall X \in B, X = \bar{F} \cup \bar{G} \cup \bar{H}$ and both conditions (1), (2), and (3) of the theorem statement hold and further $X \notin A$ (recall that $A = SM(P'_u \cup \bar{C})$).

As a consequence, either

(a) $\exists r \in P'_u \cup \bar{C} \mid r$ is false w.r.t. X, i.e. $head(r)$ is false w.r.t. X and $body(r)$ is true w.r.t. X, or
(b) $\exists X' \subset X \mid \forall r \in P'_u \cup \bar{C}, r$ is true w.r.t. X'.

In case (a), according to Definition 16, there exist $\mathscr{P}_i, r'$ s.t. $\mathscr{P}_i \in SP$, $r' \in \mathscr{P}_i$, r' is a social rule $a \leftarrow b_1, \dots b_v, s_1, \dots, s_m$ and r, according to the definition of $S'_2(\hat{\mathscr{P}}_i)$, has the form $sa_{\mathscr{P}_i} \leftarrow b^1_{\mathscr{P}_i}, \dots, b^v_{\mathscr{P}_i}, \rho(s_1)_{\mathscr{P}_i}, \dots \rho(s_m)_{\mathscr{P}_i}$.

Now, since $body(r)$ is true w.r.t. X and $X = \bar{F} \cup \bar{G} \cup \bar{H}$, for each k $(1 \leqslant k \leqslant v)$, $b^k_{\mathscr{P}_i}$ is true w.r.t. $\bar{F}$. Therefore, for each k $(1 \leqslant k \leqslant v)$, b is true for $\mathscr{P}_i$ w.r.t. $\bar{F}$.

By virtue of Lemma 1, for each l $(1 \leqslant l \leqslant m)$, s_l is true for $\mathscr{P}_i$ w.r.t. $\bar{F}$. Thus, it holds that $body(r')$ is true w.r.t. $\bar{F}$. Moreover, according to the definition of $S'_3(\mathscr{P}_i)$ (see Definition 16), $head(r')$ is false w.r.t. $\bar{F}$, since $head(r)$ is false w.r.t. $\bar{F}$.

As a consequence, the social rule $r' \in \mathscr{P}_i$ is false w.r.t. $\bar{F}$. Therefore, $\bar{F}$ is not a social model of $\mathscr{P}_1, \dots, \mathscr{P}_n$. Now, we have reached a contradiction.

In case (b), at least one of the following conditions holds, either

(α) $(X \setminus X') \cap \bar{F} \neq \emptyset$, or
(β) $(X \setminus X') \cap \bar{G} \neq \emptyset$, or
(γ) $(X \setminus X') \cap \bar{H} \neq \emptyset$.

If condition (α) occurs, according to both the hypothesis and Definition 16, then there exist $\mathscr{P}_i, r, a$ s.t $\mathscr{P}_i \in SP, r \in S'_1(\mathscr{P}_i), a_{\mathscr{P}_i} \in (X \setminus X') \cap \bar{F}$, and r has the form $a'_{\mathscr{P}_i} \leftarrow not\ a_{\mathscr{P}_i}$. Now, since $a_{\mathscr{P}_i} \in (X \setminus X') \cap \bar{F}$, it holds that $a_{\mathscr{P}_i} \in X$ and $a'_{\mathscr{P}_i} \notin X$ (otherwise, according to r, X would not be a model of $P'_u \cup \bar{C}$). Now, since $X' \subseteq X$, it holds that $a'_{\mathscr{P}_i} \notin X'$. Therefore, there exists a social rule $r \in P'_u \cup \bar{C}$ s.t. r is false w.r.t. X'. As a consequence, X' is not a model of $P'_u \cup \bar{C}$ and we have reached a contradiction. This concludes the proof of case (b), condition (α).

In case (b), if condition (β) holds, then according to both the hypothesis and Definition 16, either

(i) there exist $\mathscr{P}_i, r, a'$ s.t. $\mathscr{P}_i \in SP$, $r \in S'_1(\mathscr{P}_i)$, $a'_{\mathscr{P}_i} \in (X \setminus X') \cap \bar{G}$, and r has the form $a_{\mathscr{P}_i} \leftarrow not\ a'_{\mathscr{P}_i}$, or

(ii) there exist $\mathscr{P}_i, r, a$ s.t. $\mathscr{P}_i \in SP$, $r \in S'_3(\mathscr{P}_i)$, $a_{\mathscr{P}_i} \in \bar{X}$, $sa_{\mathscr{P}_i} \in (X \setminus X') \cap \bar{G}$, and r has the form $fail_{\mathscr{P}_i} \leftarrow not\ fail_{\mathscr{P}_i}, a_{\mathscr{P}_i}, not\ sa_{\mathscr{P}_i}$.

If item (i) is true, then r is false w.r.t. X', since $a'_{\mathscr{P}_i} \notin X'$ and $a_{\mathscr{P}_i} \notin X'$. Thus, X' is not a model of $P'_u \cup \bar{C}$ and we have reached a contradiction.

If item (ii) holds, then r is false w.r.t. X', since, according to the hypothesis, $sa_{\mathscr{P}_i} \in X$. Therefore, $a_{\mathscr{P}_i} \in X$. As a result, $sa_{\mathscr{P}_i} \notin X'$ and $a_{\mathscr{P}_i} \in X'$. Thus, X' is not a model of $P'_u \cup \bar{C}$ and we have reached a contradiction. This concludes the proof of case (b), condition (β). Now we give the proof when condition (γ) holds.

In case (b), if condition (γ) holds, then there exist $\mathscr{P}_i, h$ s.t. $\mathscr{P}_i \in SP$ and $h_{\mathscr{P}_i} \in (X \setminus X') \cap \bar{H}$. Since X' is a model of $P'_u \cup \bar{C}$, according to Definitions 12 and 21, there exists some $a_{\mathscr{P}_i}$ in $\bar{F} \cup \bar{G}$ s.t. $a_{\mathscr{P}_i} \in (X \setminus X')$. Therefore, either condition (α) or condition (β) of case (b) occurs and it is easy to see that we have reached a contradiction. Now we have concluded the proof of the theorem. □

List of Symbols and Abbreviations

$[\,]_{\mathscr{P}}$	Operator that produces a set of auxiliary atoms labeled w.r.t. $\mathscr{P}$ and used in the translation process
$\Gamma'()$	The mapping from SOLP programs to traditional logic programs
$\hat{\mathscr{P}}$	A logic program obtained from $\mathscr{P}$ after a rewriting of the tolerance rules occurring in it
$\mathrm{LP}^{\mathscr{A}}$	The nondisjunctive fragment of logic programming with aggregates supported by the DLV system
$\mathscr{SOS}(\mathscr{P}_1, \ldots, \mathscr{P}_n)$	The set of all social models of $\mathscr{P}_1, \ldots, \mathscr{P}_n$
$\mathscr{U}(\mathscr{P}_1, \ldots, \mathscr{P}_n)$	The set of all the possible combinations of autonomous fixpoints of the SOLP programs $\mathscr{P}_1, \cdots, \mathscr{P}_n$
$OKAY(\mathscr{P})$	The set of *okay* rules of the COLP program $\mathscr{P}$
$\Psi^{\mathscr{P}}(s)$	The translation of a single SC s of an SOLP program $\mathscr{P}$
$\rho(s), g(s)$	Atoms and predicates associated with the social condition s
$\sigma^n(\mathscr{P})$	The translation of a COLP program $\mathscr{P}$ into an SOLP program
$A()$	The autonomous reduction operator
$AFP(\mathscr{P})$	The set of all autonomous fixpoints of $\mathscr{P}$
$C(\mathscr{P}_1, \ldots, \mathscr{P}_n)$	The $\mathrm{LP}^{\mathscr{A}}$ program resulting from the translation of all the SCs included in the SOLP collection $\{\mathscr{P}_1, \cdots, \mathscr{P}_n\}$
$FP(\mathscr{P})$	The set of all fixpoints of $\mathscr{P}$
$IC\text{-}SOS_n$	The decision problem “individually credulous reasoning”
$IS\text{-}SOS_n$	The decision problem “individually skeptical reasoning”

$JFP(\mathscr{P}_1,\ldots,\mathscr{P}_n)$	The set of the joint fixpoints of the COLP programs $\mathscr{P}_1,\ldots,\mathscr{P}_n$
$MSC^{\mathscr{P}}$	The set of all the SCs (with depth 0) occurring in $\mathscr{P}$
$MSC^{\langle\mathscr{P},r,n\rangle}$	The set of all the SCs having a given depth n and occurring in a social rule r of an SOLP program $\mathscr{P}$
P'_u	The traditional program resulting from the translation of all the SOLP programs in an SOLP collection, where the SCs are replaced by ρ-atoms
SAT	A set of ρ-atoms and g-predicates associated with the social conditions true for a given SOLP program w.r.t. a social interpretation
$SC\text{-}SOS_n$	The decision problem “socially credulous reasoning”
$SM(\mathscr{P})$	The set of all the stable models of $\mathscr{P}$
SOS_n	The decision problem “social model existence”
$SS\text{-}SOS_n$	The decision problem “socially skeptical reasoning”
$ST_C()$	The social immediate consequence operator, applied to the SOLP collection C
$T^{\mathscr{P}}(r)$	The translation of the SCs included in a social rule r of $\mathscr{P}$
$T_{\mathscr{P}}()$	The immediate consequence operator, applied to the program $\mathscr{P}$
$TR(\mathscr{P})$	The set of tolerance rules in the program $\mathscr{P}$
$USC^{\mathscr{P}}$	The set of all the SCs (at any nesting depth) in $\mathscr{P}$
$Var(\mathscr{P})$	The set of atoms appearing in $\mathscr{P}$
$W^{\mathscr{P}}$	The translation of the SCs included in $\mathscr{P}$
(n-)SC	(n-)Social condition
COLP	Compromise logic programming
JFP	Joint fixpoint semantics
NAF	Negation as failure
SOLP	Social logic programming

References

Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P. and Torroni, P. 2004. The SOCS computational logic approach to the specification and verification of agent societies. In *Global Computing*. LNCS. Springer, Berlin/Heidelberg, 314–339.

Alferes, J. J., Leite, J. A., Pereira, L. M., Przymusinska, H. and Przymusinski, T. C. 2000. Dynamic updates of non-monotonic knowledge bases. *Journal of Logical Programming 45*(1–3), 43–70.

Alferes, J. J., Leite, J. A., Pereira, L. M., Przymusinska, H. and Przymusinski, T. C. 2002. A language for multi-dimensional updates. *Electronic Notes in Theoretical Computer Science 70* (5), 20–38.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, Cambridge, UK.

Bracciali, A., Mancarella, P., Stathis, K. and Toni, F. 2004. On modelling multi-agent systems declaratively. In *Declarative Agent Languages and Technologies DALT*, J. Leite, A. Omicini, P. Torroni and P. Yolum, Eds. Lecture Notes in Computer Science, vol. 3476. Springer, Berlin/Heidelberg, 53–68.

Brézillon, P. 1999. Context in problem solving: A survey. *The Knowledge Engineering Review 14*(1), 1–34.

Buccafurri, F. and Caminiti, G. 2005. A social semantics for multi-agent systems. In *Proceedings of 8th International Conference, LPNMR 2005,* Diamante, Italy, C. Baral, G. Greco, N. Leone, and G. Terracina, Eds. LNAI, vol. 3662. Springer-Verlag, Berlin Heidelberg, 317–329.

Buccafurri, F. and Gottlob, G. 2002. *Multiagent Compromises, Joint Fixpoints, and Stable Models.* LNCS and LNAI, vol. 2407. Springer, Berlin/Heidelberg.

Buvač, S. and Mason, I. 1993. Propositional logic of context. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, R. Fikes and W. Lehnert, Eds. American Association for Artificial Intelligence, AAAI Press, Menlo Park, California, 412–419.

Cohen, P. R. and Levesque, H. 1990. Rational interaction as the basis for communication. In *Intentions in Communication.* MIT Press, Cambridge, MA.

Cost, R. S., Finin, T. and Labrou, Y. 2001. Coordinating Agents Using ACL Conversations. In *Coordination of Internet Agents: Models, Technologies, and Applications.* 183–196.

Costantini, S. and Tocchio, A. 2002. A logic programming language for multi-agent systems. In *Proceedings of the European Conference on Logics in Artificial Intelligence, (JELIA 2002).* LNCS. Springer, Berlin/Heidelberg, 1–13.

Dell'Armi, T., Faber, W., Ielpa, G., Leone, N. and Pfeifer, G. 2003. Aggregate functions in disjunctive logic programming: Semantics, complexity, and implementation in DLV. In *IJCAI-03, Proceedings of the 18th International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, 847–852.

De Saeger, S. and Shimojima, A. 2006. Contextual reasoning in agent systems. In *Proceedings of Computational Logic in Multi-Agent Systems (CLIMA-VII)*, Hakodate, Japan.

De Vos, M. 2003. An ordered choice logic programming front-end for answer set solvers. In *Proceedings of International Joint Conference on Declarative Programming (APPIA-GULP-PRODE).* LNCS. Springer, Berlin/Heidelberg, 362–373.

De Vos, M., Crick, T., Padget, J., Brain, M., Cliffe, O. and Needham, J. 2005. LAIMA: A multi-agent platform using ordered choice logic programming. In *Proceedings of the International Workshop Declarative Agent Languages and Technologies (DALT 2005).* LNCS. Springer, Berlin/Heidelberg, 72–88.

Eiter, T., Gottlob, G. and Mannila, H. 1997. Disjunctive Datalog. *ACM Transactions on Database Systems 22*(3), 364–418.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *5th Conference on Logic Programming*. MIT Press, Cambridge, MA, 1070–1080.

Gelfond, M. and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computation 9*(3/4), 365–386.

Ghidini, C. and Giunchiglia, F. 2001. Local models semantics, or contextual reasoning = locality + compatibility. *Artificial Intelligence 127*(2), 221–259.

Konolige, K. 1984. A Deduction Model of Belief and its Logics. Ph.D. Thesis, Stanford University CA.

Leite, J. A., Alferes, J. J. and Pereira, L. M. 2002. MINERVA: A dynamic logic programming agent architecture. In *Proceedings of ATAL-2001.* LNAI, Springer, Berlin/Heidelberg, 141–157.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S. and Scarcello, F. 2002. The DLV System for Knowledge Representation and Reasoning. *ArXiv Computer Science e-prints*, 11004–+.

Mascardi, V., Martelli, M. and Sterling, L. 2004. Logic-based specification languages for intelligent software agents. *Theory and Practice of Logic Programming 4*(4), 429–494.

Mayfield, J., Yannis, L. and Finin, T. 1995. Evaluation of kqml as an agent communication language. In *Proceedings of the 2nd International Workshop on Agent Theories, Architectures,* *and Languages (ATAL'95)*, M. J. P. Wooldridge, M. and M. Tambe, Eds. Number 1037 in LNAI. Springer-Verlag, Berlin/Heidelberg, 347–360.

McCarthy, J. 1993. Notes on formalizing contexts. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, R. Bajcsy, Ed. Morgan Kaufmann, San Mateo, California, 555–560.

Rao, A. S. 1996. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In *Agents Breaking Away*, W. Van de Velde and J. W. Perram, Eds. vol. 1038. LNAI. Springer-Verlag, Berlin/Heidelberg, 42–55.

Rao, A. S. and Georgeff, M. 1995. Bdi agents: From theory to practice. In *Proceedings of the 1st International Conference on Multi Agent Systems (ICMAS'95)*, V. Lesser, Ed. AAAI Press, Cambridge, MA, 312–319.

Satoh, K. and Yamamoto, K. 2002. Speculative computation with multi-agent belief revision. In *The First International Joint Conference on Autonomous Agents & Multiagent Systems*. ACM Press, New York, 897–904.

Serafini, L. and Bouquet, P. 2004. Comparing formal theories of context in ai. *Artificial Intelligence 155*(1–2), 41–67.

Subrahmanian, V., Bonatti, P., Dix, J., Eiter, T., Kraus, S., Ozcan, F. and Ross, R. 2000. *Heterogeneous Agent Systems*. MIT Press/AAAI Press, Cambridge, MA.

van der Hoek, W. and Wooldrige, W. 2003. Towards a logic of rational agency. *Logic Journal of the IGPL 11*(2), 135–159.

Wooldridge, M. 2000. *Reasoning about Rational Agents*. Intelligent Robots and Autonomous Agents. MIT Press, Cambridge, MA.

Wooldridge, M. and Jennings, N. R. 1995. Intelligent agents: Theory and practice. *The Knowledge Engineering Review 2*(10), 115–152.