Hierarchical Decision Making in Multi-Agent Systems using Answer Set Programming

Davy Van Nieuwenborgh[1⋆], Marina De Vos[2], Stijn Heymans[1], and Dirk Vermeir[1]

[1] Dept. of Computer Science
Vrije Universiteit Brussel, VUB
Pleinlaan 2, B1050 Brussels, Belgium
{dvnieuwe,sheymans,dvermeir}@vub.ac.be
[2] Dept. of Computer Science
University of Bath
Bath, BA2 7AY, UK
mdv@cs.bath.ac.uk

Abstract. We present a multi-agent formalism based on extended answer set programming. The system consists of independent agents connected via a communication channel, where knowledge and beliefs of each agent are represented by a logic program. When presented with an input set of literals from its predecessor, an agent computes its output as an extended answer set of its program enriched with the input, carefully eliminating contradictions that might occur.
It turns out that while individual agents are rather simple, the interaction strategy makes the system quite expressive: e.g. the membership problem for a sequence of n agents is Σ_n^P-complete. This makes the formalism suitable for modelling complex applications of MAS, for example cooperative diagnosis. Furthermore, such systems can be realized by implementing an appropriate control strategy on top of existing solvers such as DLV and SMODELS.

1 Introduction

In *answer set programming* ([22]) a logic program is used to intuitively describe the requirements that must be fulfilled by the solutions of a certain problem. The answer sets of the program, usually defined through (a variant/extension of) the stable model semantics [18], then correspond to the solutions of the problem. This technique has been successfully applied in problem areas such as planning [12, 22], configuration and verification [28], diagnosis [11, 34] and game theory [9]. In the context of multi-agent systems, answer set programming has been used in [1, 4, 10]. While [1] and [4] use the basic answer set semantics to represent an agent's domain knowledge, [10] applies an extension of the semantics incorporating preferences among choices in a program.

The traditional answer set semantics, even in the absence of constraints, is not universal, i.e. some programs may not have any answer set at all. While natural, this poses a problem in cases where there are no exact solutions, but one would appreciate to obtaine approximate ones, even if they violate some rules. For example, it is not acceptable

⋆ Supported by the Flemish Fund for Scientific Research (FWO-Vlaanderen).

that an airplane's auto-pilot agent fails to work just because it has some contradictory readings regarding the outside temperature. To achieve this, the extended answer set semantics ([33]) allows problematic rules to be *defeated*: the rules $a \leftarrow$, $b \leftarrow$ and $\neg a \leftarrow b$ are clearly inconsistent and have no classical answer set, while both $\{a, b\}$ and $\{\neg a, b\}$ will be recognized as extended answer sets. In $\{a, b\}$, $\neg a \leftarrow b$ is defeated by $a \leftarrow$, while in $\{\neg a, b\}$, $a \leftarrow$ is defeated by $\neg a \leftarrow b$.

In this paper we use the extended answer set semantics to model the knowledge and beliefs of a single agent. Each agent reasons over two languages, one public and one private. This allows agents to dynamically decide which information they wish to share with others, with only public information being made available. Agents may then cooperate to select among the various possible solutions (extended answer sets) that are presented to them. In the case that an agent, using the extended answer set semantics, has a number of (approximate) solutions to a certain problem, it can rely upon other agents to sort out which solutions are the better ones. In the absence of any extended answer sets, the agent relies completely on the information received from the others, e.g., when a company has to make up an emergency evacuation plan for a building, one of the employees will make up all strategies that could be implemented for that building. However, as she is probably not aware of all current regulations about such strategies, her solutions are forwarded to the emergency services, who will only select those plans that are conforming to all legal requirements. These legal candidate plans are then presented to the firm's management to select an optimal one (e.g. the cheapest) for implementation.

To deal with problems like the one described above, we propose a multi-agent framework that is capable of modelling hierarchical decision problems. To this end, we consider a sequence of agents $A_1 \ldots A_n$, each having their private knowledge described by a logic program. Intuitively, an agent A_i communicates a solution she finds acceptable to the next agent A_{i+1} in the hierarchy. For such an A_i-acceptable solution, A_{i+1} computes a solution S that adds her knowledge to the given information. Provided that this new knowledge does not conflict with the information she received from her predecessor A_i, she passes this solution to the following agent in line, i.e. A_{i+2}. In case agent A_{i+1} is unable to provide any solutions of her own, she will simply pass on information she obtained from the previous agents higher up in the hierarchy. When her solution S conflicts with the solution offered by her predecessors, she sends S for verification to her predecessor A_i. If A_i is able to find another possible solution T that is consistent with S, the communication from A_i to A_{i+1} starts over again with T as a new input. In the case that none of the solutions of A_{i+1} survive the verification step, A_{i+1} has no other option than accepting the input from A_i and send it to A_{i+2}.

It turns out that, although the agents are relatively simple in complexity terms, such sequences of agents are rather expressive. More specifically, we show that arbitrary complete problems of the polynomial hierarchy can be solved by such agent systems, which make them suitable for encoding complex applications.

Computing the extended answer set semantics is located at the first level of the polynomial hierarchy. Problems located at this first level can be directly solved using the DLV [16] and SMODELS [26] answer set solvers. On the second level, only DLV remains to perform the job directly. However, by using a "guess and check" fixpoint procedure, SMODELS can indirectly be used to solve problems at the second level [2, 14, 37]. Beyond the second level, there are still some interesting problems, such as the most expressive forms of diagnostic reasoning, i.e. subset-minimal diagnosis on disjunctive system descriptions [11] or preference-based diagnosis on ordered theories [34]. These are located at the third level of the polynomial hierarchy, together with sequences of weak constraints[3] on disjunctive programs. For these problems, and problems located even higher in the polynomial hierarchy, no direct computational vehicle is available. The framework presented in this paper provides a means to effectively compute solutions for such problems with each agent using SMODELS or DLV to compute better solutions combined with an appropriate control strategy for the communication.

The remainder of the paper is organized as follows. In Section 2, we review the extended answer set semantics. Section 3 presents the definitions for hierarchical agents and agent systems. Section 4 discusses the complexity of the proposed semantics, while Section 5 compares it with related approaches from the literature. Finally, we conclude with directions for further research in Section 6.

2 Extended Answer Sets

In this section we provide a short overview of extended answer set semantics for simple logic programs [32]. A *literal* is an atom a or a negated atom $\neg a$. For a set of literals X, we take $\neg X = \{\neg l \mid l \in X\}$ where $\neg\neg a$ is a. When $X \cap \neg X = \emptyset$ we say X is *consistent*. A *simple logic program* (SLP) is a finite set of *simple rules*[4] of the form $\alpha \leftarrow \beta$ with $\alpha \cup \beta$ a set of literals and $|\alpha| \leq 1$. If $\alpha = \emptyset$, we call the rule a *constraint*. The set α is the *head* of the rule while β is called the *body*.

For a program P, the set of all atoms that appear in the program (possibly negated) is called the *Herbrand Base* $\mathcal{B}_P$. The set of all literals that can be formed using $\mathcal{B}_P$, denoted by $\mathcal{L}_P$, is defined by $\mathcal{L}_P = \mathcal{B}_P \cup \neg\mathcal{B}_P$.

Any consistent subset $I \subseteq \mathcal{L}_P$ is called an *interpretation* of P. A rule $r = a \leftarrow \beta \in P$ is *satisfied* by an interpretation I, denoted $I \models r$, if $a \in I$ whenever $\beta \subseteq I$, i.e. if r is *applicable* ($\beta \subseteq I$), then it must be *applied* ($\beta \cup \{a\} \subseteq I$). On the other hand, a constraint $\leftarrow \beta$ is satisfied if $\beta \not\subseteq I$, i.e. the constraint is not applicable. The rule r is said to be *defeated* w.r.t. I iff there exists an applied *competing rule* $\neg a \leftarrow \beta' \in P$. We use $P_I \subseteq P$ to denote the *reduct* of P w.r.t. I, i.e. $P_I = \{r \in P \mid I \models r\}$, the set of rules satisfied by I.

If an interpretation I satisfies all rules in P, i.e. $P_I = P$, I is called a *model* of P. A model I is a minimal model or *answer set* of P iff no other model J of P exists such that $J \subset I$. An *extended answer set* of P is any interpretation I such that I is an answer set of P_I and each unsatisfied rule in $P \backslash P_I$ is defeated. The set of all extended answer sets of a program P is denoted by $\mathcal{AS}(P)$.

[3] A weak constraint is a constraint that is "desirable" but may be violated if there are no other options.
[4] As usual, we assume that programs have already been grounded, i.e. the variables have been replaced by constants.

Example 1. Consider the following SLP P about diabetes.

$$\begin{array}{rcc} hypoglycemia \leftarrow & sugar \leftarrow hypoglycemia & coke \leftarrow sugar \\ diabetes \leftarrow & \neg sugar \leftarrow diabetes & diet\_coke \leftarrow \neg sugar \end{array}$$

Clearly, while this program has no traditional answer sets, it does have two extended answer sets $I = \{diabetes, hypoglycemia, sugar, coke\}$ and $J = \{diabetes, diet\_coke, hypoglycemia, \neg sugar\}$.

Note that the extended answer set semantics is universal for simple programs containing no constraints [32]. This is not the case for general, constraint allowing, simple programs, due to the fact that constraints cannot be defeated.

3 Hierarchical Agents

If humans want to share information or have discussions in an effective manner, it is only normal that they use the same language; without, it would be impossible to establish any communication. So it is only natural that we assume that all agents in our framework "speak the same language" which we denote as $\mathcal{AL}$. Modelling an agent's knowledge and beliefs, it might not always be a good idea to pass on the entire answer set, e.g., a manager is certainly not going to tell her employee that she cannot have a meeting on Monday because she wants to have an extended weekend in Paris. Instead she will simply say that Monday is out of the question. To allow this we need to perform some filtering on the answer set before it is passed to the next agent. For this reason, we consider agents that use two languages: a public language $\mathcal{AL}$ used for communication and a private language $\mathcal{AL}'$ for private reasoning purposes. The latter allows the manager in our example to tell her employee she cannot have the meeting on Monday, without giving her the underlying reason that she is in Paris for a trip. On the other hand, if it is a business trip, she could choose to communicate the reason. Information received from other agents will be assumed private by default. If it needs to be passed one simply adds a rule $l \leftarrow l'$ for each literal that could be received from the other agent. Summarised, an agent will receive input in the language $\mathcal{L}_{\mathcal{AL}}$, do some reasoning with a program over $\mathcal{L}_{\mathcal{AL}} \cup \mathcal{L}_{\mathcal{AL}'}$ and will only communicate the part over $\mathcal{L}_{\mathcal{AL}}$ to the other agents. In this context, we use l' to denote the private version of the literal $l \in \mathcal{L}_{\mathcal{AL}}$ in $\mathcal{L}_{\mathcal{AL}'}$ and we have for $l' \in \mathcal{L}_{\mathcal{AL}'}$ that $l'' = l \in \mathcal{L}_{\mathcal{AL}}$. We extend the notation as usual to a set $X \subseteq \mathcal{L}_{\mathcal{AL}} \cup \mathcal{L}_{\mathcal{AL}'}$, i.e. $X' = \{l' \mid l \in X\}$.

Definition 1. *For an agent language $\mathcal{AL}$, a* ***hierarchical agent*** *A is a SLP such that $\mathcal{B}_A \subseteq \mathcal{AL} \cup \mathcal{AL}'$. For such an agent A and a set of literals $I \subseteq \mathcal{L}_{\mathcal{AL}}$, the* ***agent input****, we use $A(I)$ to denote the SLP $A \cup \{l' \leftarrow \ \mid l \in I\}$.*

An interpretation $S \subseteq \mathcal{L}_{\mathcal{AL}}$ is an ***agent answer set*** *w.r.t. the agent input I if*
– *$S = M \cap \mathcal{L}_{\mathcal{AL}}$ with $M \in \mathcal{AS}(A(I))$, or*
– *$S = I$ when $\mathcal{AS}(A(I)) = \emptyset$.*

We use $\mathcal{AS}(A, I)$ to denote the set of all agent answer sets of A w.r.t. input I.

The first condition of the agent answer set definition ensures that the agent only communicates public information. The second condition makes that the agent answer set semantics is universal. In case our agent cannot produce an answer set, because of constraints being violated, she will assume the input as the solution. This makes sense in the context of hierarchical agents, as orders from a superior should be obeyed even if they are in conflict with your own beliefs. As an employee, you should reschedule your meeting with others if your boss can only make it on a time you already were booked up.

Example 2. Take $\mathcal{AL} = \{hypoglycemia, diabetes, sugar, coke, diet\_coke\}$ and consider the following diabetes agent A.

$$\begin{array}{rr} sugar' \leftarrow hypoglycemia' & \neg sugar' \leftarrow diabetes' \\ diet\_coke \leftarrow \neg sugar' & coke \leftarrow sugar' \end{array}$$

Intuitively, the above agent is set up to use information from a doctor agent concerning hypoglycemia and diabetes to decide if a patient needs to have diet coke or normal coke. In order to do so, she derives if the patient needs sugar or should not have sugar. The patient only needs to be told that she can have either a diet coke or a normal coke, hence diet coke and coke are the only literals in the public language.

Let $I_1 = \emptyset$, $I_2 = \{diabetes\}$ and $I_3 = \{hypoglycemia\}$ be three agent inputs. One can check that A has only one agent answer set w.r.t. I_1 which is $S_1 = \emptyset$. Similar, feeding both I_2 and I_3 as input to A results in a single agent answer set, i.e. $S_2 = \{diet\_coke\}$ and $S_3 = \{coke\}$ respectively.

As mentioned before, information an agent receives can be easily made public by adding a rule $l \leftarrow l'$ for each literal one wants to make public. Depending on the agent, a large number of these literals have to be made public. To shorten the programs, we introduce the short hand **pass**(S) for $\{l \leftarrow l' \mid l \in S\}$, with $S \subseteq \mathcal{AL}$ the set of literals that need to be made public if derived in the private part.

Using a combination of public and private information, it is possible to easily encode that for example certain input information should be considered more important than the agent's own knowledge or vice versa.

Example 3. Consider the following employee agent A_1:

$$\begin{array}{lrr} \textbf{pass}(\{rise, overworked\}) & & \\ & overworked \leftarrow & boss\_hated' \leftarrow overworked \\ & happy \leftarrow \neg boss\_hated', rise' & \neg boss\_hated' \leftarrow rise' \end{array}$$

Obviously the agent will never publicly admit hating the boss. Given $\{rise\}$ as input, the agent produces two answer sets: $\{rise, overworked\}$ and $\{rise, overworked, happy\}$.

Now that we have defined a single hierarchical agent and the semantics that comes with it, we can start to connect them. As mentioned previously, we are interested in multi-agent systems where some agents have more authority than others, yet require information from others in order to make correct decisions. In the introduction, we discussed the situation of a company that needs to implement an emergency evacuation plan. Although a manager needs to approve the emergency plan, she does not need to verify legal issues or draw up the plans herself. She will stipulate the requirements that need to be fulfilled for her approval. So, in this case we have the employee being on top of the hierarchy generating all possible plans, followed by the legal office rejecting those plans which are not safe. Finally, these plans will be matched against the criteria set out by the manager. Since a plan is needed, she will be unable to reject them all.

We have a different situation when a head of department needs to arrange a meeting with her staff. Obviously she will allow her staff to have a say in the organisation, but at the end of the day her diary will take precedence over that of her staff. Here the head of department will be on top of the hierarchy to generate all possible dates she can have the meeting, which can then be verified by her staff.

The above two examples demonstrate that there can be a difference between the agent hierarchy and the hierarchy of the people/things modelled by the agents. The agent with the greatest power is the one generating all the candidate models. The effect of the lower agents is proportional to their level in the hierarchy. In this paper, we restrict to linearly connected agents, since such systems are already capable of representing the most common forms of hierarchy.

Formally, a *hierarchical agent system (HAS)* is a linear sequence of hierarchical agents $A = (A_1, \ldots, A_n)$, where A_1 is the source agent, i.e. the agent that starts all communication. For a HAS A, we refer to the i-th agent as A_i, while we use $A_{<i}$ to denote the HAS consisting of the predecessors of A_i, i.e. $A_{<i} = (A_1, \ldots, A_{i-1})$.

We assume for our theoretical model that agents are fully aware of the agents that they can communicate with (as the communication structure is fixed) and that they can communicate by passing sets of literals over communication channels. When put to practice in an open multi-agent environment, an agent would first engage in establishing a community and the appropriate hierarchy before collaborating on establishing a consensus on the answer sets. Furthermore, one would expect the set of literals encapsulated in a communication protocol.

Each agent in a HAS is a separate entity with its own reasoning skills, knowledge and beliefs. Each agent has the right to remove or add information to the input as she sees fit. To reflect this, we introduce an *interpretation* for a HAS $A = (A_1, \ldots, A_n)$ as a sequence of interpretations $I = (I_1, \ldots, I_n)$, one for each agent, denoting the public knowledge of each individual agent. For interpretations, we introduce the same notation I_i and $I_{<i}$ as we did for hierarchical agent systems. An interpretation I is consistent iff $\bigcup_{1 \leq i \leq n} I_i$ is consistent. Given a sequence I and a set S, we will write (I, S) to denote the new sequence obtained from concatenating I and S.

Example 4. Consider the HAS $A = (A_1, A_2)$ with $A_1 = \{meeting \leftarrow\}$ and $A_2 = \{out\_of\_office \leftarrow meeting'\}$. Then, $(\{meeting\}, \{out\_of\_office\})$ is an interpretation.

The solutions of such a hierarchical agent system, called hierarchical answer sets, are defined inductively. For a HAS A, we will use $\mathcal{AG}(A)$ to denote the set of all hierarchical answer sets of A.

Definition 2. *Let $\mathcal{AL}$ be an agent language.*

– *A **hierarchical answer set** of a HAS $A = (A_1)$ is a consistent interpretation S such that $S_1 \in \mathcal{AS}(A_1, \emptyset)$.*

– *A **hierarchical answer set** of a HAS $A = (A_1, \ldots, A_n)$ is a consistent interpretation S such that $S_{<n}$ is a hierarchical answer set of $A_{<n}$, i.e. $S_{<n} \in \mathcal{AG}(A_{<n})$, and*
  1. *$S_n \in \mathcal{AS}(A_n, S_{n-1})$; or*
  2. *$S_n = S_{n-1}$ iff $\forall S' \in \mathcal{AG}(A_{<n}) \cdot \forall T \in \mathcal{AS}(A_n, S_{n-1}) \cdot (S', T)$ inconsistent.*

The case of a single agent HAS is simple: hierarchical answer sets equal the agent's agent answer sets with empty input. The two conditions of the general case are the encoding of the principle that an agent either has to be able to augment the input in a consistent manner (condition 1) or convince itself that all the alternatives it can propose are inconsistent with solutions that are acceptable by its predecessors. In that case, the input will be accepted (condition 2). If not, the candidate will be rejected.

Example 5. Consider the following simple HAS $A = (A_1, A_2, A_3)$ with:
– the general director A_1 of a company containing the following rules[5]:

$$monday \oplus tuesday \oplus friday \leftarrow \qquad \neg wednesday \leftarrow \qquad \neg thursday \leftarrow$$

– the head of research A_2 containing the rules:

$$monday \oplus thursday \leftarrow \qquad \neg tuesday \leftarrow \qquad \neg wednesday \leftarrow \qquad \neg friday \leftarrow$$

– the project manager A_3 containing the rules:

$$friday \oplus wednesday \leftarrow \qquad \neg monday \leftarrow \qquad \neg tuesday \leftarrow \qquad \neg thursday \leftarrow$$

who attempts to arrange a meeting. The director agent produces three possible hierarchical answer sets for the HAS (A_1), i.e.
– $(M_1) = (\{monday, \neg tuesday, \neg wednesday, \neg thursday, \neg friday\})$
– $(M_2) = (\{\neg monday, tuesday, \neg wednesday, \neg thursday, \neg friday\})$
– $(M_3) = (\{\neg monday, \neg tuesday, \neg wednesday, \neg thursday, friday\})$

Let us now consider $A_{<3} = (A_1, A_2)$. When we feed A_2 with M_1, we notice that M_1 is accepted. This means that (M_1, M_1) is a hierarchical answer set for $A_{<3}$. Any other answer set from A_2 with input M_1 leads to contradiction. When we use M_2 as input we have that $M_1 \in \mathcal{AS}(A_2, M_2,)$ is clearly inconsistent with M_2, but which is consistent with an acceptable solution of the predecessors, i.e. (M_1). This implies that there is no hierarchical answer with M_2 as input for A_2. The same is true when M_3 is used as input. As a result, we have $\mathcal{AG}(A_{<3}) = \{(M_1, M_1)\}$.

Now that we have the hierarchical answer sets for $A_{<3}$, we can define those of A. When we compute the answer sets of A_3 with M_1 as input, we obtain two answer sets: one assuming friday to be true and the other wednesday to be true. Both are inconsistent with (M_1, M_1), so our project manager has no other option than to conform to M_1 herself, resulting in $\mathcal{AG}(A) = \{(M_1, M_1, M_1)\}$.

Now consider the rearranged HAS $B = (A_1, A_3, A_2)$, e.g. because A_3 has prior arrangements with clients who do not appreciate changes to their schedule. This change would result in a different hierarchical answer set, namely $\mathcal{AG}(B) = \{(M_3, M_3, M_3)\}$.

[5] In the following we will use rules of the form $a \oplus b \oplus c \leftarrow$ to denote the set of rules $\{a \leftarrow\ ;\ b \leftarrow\ ;\ c \leftarrow\ ;\ \neg a \leftarrow\ ;\ \neg b \leftarrow\ ;\ \neg c \leftarrow\ ;\ \leftarrow a, b\ ;\ \leftarrow a, c\ ;\ \leftarrow b, c\ ;\ \leftarrow \neg a, \neg b, \neg c\}$, i.e. an exclusive choice between a, b and c.

Although we request that hierarchical answer sets are consistent, this does not mean that internal inconsistencies cannot appear. Further, the system also allows for cheating and/or lying.

Example 6. Consider the following HAS $A = (A_1, A_2)$ with $A_1 = \{a \leftarrow\}$ and $A_2 = \{b \leftarrow; \neg a' \leftarrow b; c \leftarrow \neg a'; a \leftarrow a'\}$. This HAS produces two hierarchical answer sets: $(\{a\}, \{a, b\})$ and $(\{a\}, \{b, c\})$. In the latter case, the agent A_2 knows that there would be a contradiction if she would admit $\neg a$, so she decides to pretend she does not know anything about $\neg a$ and only states the implication of $\neg a'$, i.e. the conclusion c.

Example 7. Consider the job selection procedure of a company. The first agent A_1 corresponds with the possible profiles of the applicants. Thus, each agent answer set of the agent below corresponds with a possible applicant's profile.

$$male \oplus female \leftarrow \qquad old \oplus young \leftarrow \qquad experienced \oplus inexperienced \leftarrow$$

The decision which applicant gets the job goes through a chain of decision makers. First, the agent A_2 of the human resources department implements company policy which stipulates that experienced persons should be preferred over inexperienced ones. Therefore, the agent passes through all of its input, except when it encounters a profile containing *inexperienced*, which it changes to *experienced*, intuitively implementing that an applicant with the same profile but *experienced* instead of *inexperienced*, would be preferable. Further, the department is convinced that younger employees are ambitious.

$$\begin{array}{c}
\mathbf{pass}(\{male, female, old, young, experienced\}) \\
\mathbf{pass}(\{\neg male, \neg female, \neg old, \neg young, \neg inexperienced\}) \\
experienced \leftarrow inexperienced' \\
\neg inexperienced \leftarrow inexperienced' \\
ambitious \leftarrow young'
\end{array}$$

On the next level of the decision chain, the financial department reviews the remaining candidates. As young and inexperienced persons tend to cost less, it has a strong desire to hire such candidates, which is implemented in the following agent A_3.

$$\begin{array}{c}
\mathbf{pass}(\{male, female, young, inexperienced\}) \\
\mathbf{pass}(\{\neg male, \neg female, \neg old, \neg experienced\})
\end{array}$$

$$\begin{array}{rlrl}
inexperienced & \leftarrow young', experienced' & \neg experienced & \leftarrow young', experienced' \\
young & \leftarrow young', experienced' & \neg old & \leftarrow young', experienced' \\
young & \leftarrow old', inexperienced' & \neg old & \leftarrow old', inexperienced' \\
inexperienced & \leftarrow old', inexperienced' & \neg experienced & \leftarrow old', inexperienced' \\
inexperienced & \leftarrow old', experienced' & \neg inexperienced & \leftarrow old', experienced' \\
experienced' & \leftarrow old, experienced & \neg experienced' & \leftarrow old, experienced \\
young & \leftarrow old', experienced' & \neg young & \leftarrow old', experienced' \\
old & \leftarrow old', experienced' & \neg old & \leftarrow old', experienced' \\
 & \leftarrow old, experienced & cheaper & \leftarrow inexperienced \\
 & & cheaper & \leftarrow young
\end{array}$$

Intuitively, this agent handles the four possible cases: when the input profile is from a young and inexperienced person, nothing will be changed, indicating that the input cannot be improved. On the other hand, if only one of the properties is not as desired, e.g. *young* and *experienced*, then the only improvement would be a profile containing both *young* and *inexperienced*. Finally, a profile containing *old* and *experienced* has three possible improvements: the contradictory rules together with the constraint ensure that the agent answer sets proposed by A_3 will contain *young* or *inexperienced*, or both.

Finally, the management has the final call in the selection procedure. As the current team of employees is largely male, the management prefers the new worker to be a woman, as described by the next agent A_4, which is similar to A_2.

$$\textbf{pass}(\mathcal{AL} \setminus \{male, \neg female, ambitious, cheaper\})$$
$$female \leftarrow male' \qquad \neg male \leftarrow \neg female' \qquad \leftarrow female'$$

One can check that the system (A_1) has eight hierarchical answer sets, among them are

$$(M_1) = (\{experienced, \neg inexperienced, male, \neg female, young, \neg old\}) \ ,$$
$$(M_2) = (\{experienced, \neg inexperienced, male, \neg female, old, \neg young\}) \ ,$$
$$(M_3) = (\{experienced, \neg inexperienced, female, \neg male, young, \neg old\}) \ ,$$
$$(M_4) = (\{experienced, \neg inexperienced, female, \neg male, old, \neg young\}) \ ,$$
$$(M_5) = (\{inexperienced, \neg experienced, female, \neg male, young, \neg old\}) \ .$$

However, only four of these will survive agent A_2, i.e. $\mathcal{AG}((A_1, A_2)) = \{(M_1, M_1 \cup \{ambitious\}), (M_2, M_2), (M_3, M_3 \cup \{ambitious\}), (M_4, M_4)\}$, which fits the human resource policy to drop inexperienced people. Feeding M_5 as input to A_2 yields one agent answer set $M_3 \cup \{ambitious\}$, which is consistent with $(M_3) \in \mathcal{AG}((A_1))$, making (M_5, M_5) unacceptable as a solution for the system. Similarly, when agent A_3 is taken into account, only $(M_1, M_1 \cup \{ambitious\}, M_1 \cup \{cheaper\})$ and $(M_3, M_3 \cup \{ambitious\}, M_3 \cup \{cheaper\})$ are contained in $\mathcal{AG}((A_1, A_2, A_3))$. Considering the last agent A_4, the HAS (A_1, A_2, A_3, A_4) yields a single hierarchical answer set,

$$(M_3, M_3 \cup \{ambitious\}, M_3 \cup \{cheaper\}, M_3) \ ,$$

which fits our intuition that, if possible, a woman should get the job.

Definitions 1 and 2 ensure that each single agent HAS has at least one hierarchical answer set. Extending such a HAS with other agents, implies those successors can either augment one of these solutions in a consistent manner or simply accept them. This implies that an arbitrary HAS always provide at least one solution, i.e. the hierarchical answer set semantics is universal.

Theorem 1. *Let A be a HAS . Then, $\mathcal{AG}(A) \neq \emptyset$.*

The hierarchical answer set semantics is monotonic.

Theorem 2. *Let $A = (A_1, \ldots, A_n)$ and $B = (B_1, \ldots, B_m)$ with $m > n$ such that $B_{<(n+1)} = A$. Then, $\forall S \in \mathcal{AG}(B) \cdot S_{<(n+1)} \in \mathcal{AG}(A)$.*

4 Complexity

We briefly recall some relevant notions of complexity theory (see [23] for an introduction). The class $\mathcal{P}$ ($\mathcal{NP}$) represents the problems that are deterministically (nondeterministically) decidable in polynomial time, while $co\mathcal{NP}$ contains the problems whose complements are in $\mathcal{NP}$.

The polynomial hierarchy, denoted $\mathcal{PH}$, is made up of three classes of problems, i.e. Δ_k^P, Σ_k^P and Π_k^P, $k \geq 0$, which are defined as $\Delta_0^P = \Sigma_0^P = \Pi_0^P = \mathcal{P}$; $\Delta_{k+1}^P = \mathcal{P}^{\Sigma_k^P}$; $\Sigma_{k+1}^P = \mathcal{NP}^{\Sigma_k^P}$ and $\Pi_{k+1}^P = co\Sigma_{k+1}^P$.

The class $\mathcal{P}^{\Sigma_k^P}$ ($\mathcal{NP}^{\Sigma_k^P}$) represents the problems decidable in deterministic (nondeterministic) polynomial time using an oracle for problems in Σ_k^P, where an oracle is a subroutine capable of solving Σ_k^P problems in unit time. Note that $\Delta_1^P = P$, $\Sigma_1^P = \mathcal{NP}$ and $\Pi_1^P = co\mathcal{NP}$. Further, it is obvious that $\Sigma_k^P \subseteq \Sigma_k^P \cup \Pi_k^P \subseteq \Delta_{k+1}^P \subseteq \Sigma_{k+1}^P$, but for $k \geq 1$ any equality is considered unlikely. Further, the class $\mathcal{PH}$ is defined by $\mathcal{PH} = \bigcup_{k=0}^{\infty} \Sigma_k^P$.

A decision problem D is called complete for a complexity class C if both D is in C and D is hard for C. Showing that D is hard is normally done by reducing a known complete decision problem into the decision problem D. For the classes Σ_k^P and Π_k^P with $k > 0$ a known complete, under polynomial time transformations, the problem is checking whether a quantified boolean formula (QBF) ϕ is valid. Note that this does not hold for the class $\mathcal{PH}$ for which no complete problem is known unless $\mathcal{P} = \mathcal{NP}$.

Quantified boolean formulas are expressions of the form $Q_1 X_1 Q_2 X_2 \ldots Q_k X_k \cdot G$, where $k \geq 1$, G is a Boolean expression over the atoms of the pairwise nonempty disjoint sets of variables $X_1, \ldots, X_k$ and the Q_i's, for $i = 1, \ldots, k$ are alternating quantifiers from $\{\exists, \forall\}$. When $Q_1 = \exists$, the QBF is k-existential, when $Q_1 = \forall$ we say it is k-universal. We use $QBF_{k,\exists}$ ($QBF_{k,\forall}$) to denote the set of all valid k-existential (k-universal) QBFs. Deciding, for a given k-existential (k-universal) QBF ϕ, whether $\phi \in QBF_{k,\exists}$ ($\phi \in QBF_{k,\forall}$) is a Σ_k^P-complete (Π_k^P-complete) problem.

The following results shed some light on the complexity of the hierarchical answer set semantics for hierarchical agent systems. Due to space restrictions we do not provide the actual proofs, but they can be found in the technical report [30]. However, we do provide the intuition behind the construction of the hardness part by means of an example.

Theorem 3. *The problem of deciding, given a HAS $(A_i)_{i=1,\ldots,n}$, with n fixed, and a literal $l \in \mathcal{L}_{\mathcal{AL}}$, whether there exists a hierarchical answer set I containing l is Σ_n^P-complete. On the other hand, deciding whether every hierarchical answer set contains l is Π_n^P-complete.*

Proof. To prove hardness, we provide a reduction of deciding validity of QBFs by means of a HAS .

Let $\phi = \exists X_1 \forall X_2 \ldots Q X_n \cdot G \in QBF_{n,\exists}$, where $Q = \forall$ if n is even and $Q = \exists$ otherwise. We assume, without loss of generality [29], that G is in disjunctive normal form, i.e. $G = \vee_{c \in C} C$ where C is a set of sets of literals over $X_1 \cup \ldots \cup X_n$ and each $c \in C$ has to be read as a conjunction.

In what follows, we will use $P_\forall^i$ to denote the program containing the rules

- **pass**$(\{x, \neg x \mid x \in X_j \wedge 1 \leq j < i\})$,
- $\{x' \leftarrow\ ;\ \neg x' \leftarrow\ \mid x \in X_j \wedge i \leq j \leq n\}$,
- $\{sat' \leftarrow c' \mid c \in C\}$,
- $\{\ \leftarrow sat'\ ;\ \neg sat \leftarrow\ ;\ \leftarrow \neg sat\}$.

Similarly, we use $P_\exists^i$ to denote the program

- **pass**$(\{x, \neg x \mid x \in X_j \wedge 1 \leq j < i\})$,
- $\{x' \leftarrow\ ;\ \neg x' \leftarrow\ \mid x \in X_j \wedge i \leq j \leq n\}$,
- $\{sat' \leftarrow c' \mid c \in C\}$,
- $\{\neg sat' \leftarrow\ ;\ \leftarrow \neg sat'\ ;\ \leftarrow sat\}$.

The HAS $A_\phi = (A_1, \ldots, A_n)$ corresponding to ϕ is defined by the following hierarchical agents:

- A_1 contains the rules $\{x' \leftarrow\ ;\ \neg x' \leftarrow\ \mid x \in X_j \wedge 1 \leq j \leq n\}$ and $\{sat' \leftarrow c' \mid c \in C\}$;
- if n is even, then $A_i = P_\forall^{n+2-i}$ when i even and $A_i = P_\exists^{n+2-i}$ when $i > 1$ odd;
- if n is odd, then $A_i = P_\exists^{n+2-i}$ when i even and $A_i = P_\forall^{n+2-i}$ when $i > 1$ odd.

Obviously, the above construction can be done in polynomial time. Intuitively, the hierarchical agent A_1 has agent answer sets for every possible combination of the X_i's and if such a combination makes G valid, then the corresponding agent answer set also contains the atom sat. The intuition behind the hierarchical agent $P_\forall^i$ is that it tries to disprove, for the received input, the validity of the corresponding $\forall$, i.e. for a given input combination over the X_j's making G satisfied, the hierarchical agent $P_\forall^i$ will try to find a combination, keeping the X_j's with $j < i$ fixed, making G false. On the other hand, the hierarchical agent $P_\exists^i$ will try to prove the validity of the corresponding $\exists$, i.e. for a given combination making G false it will try to compute a combination, keeping the X_j's with $j < i$ fixed, making G satisfied.

Instead of giving the formal proof for the above construction, we illustrate, for clarity, the construction and the working of the HAS A_ϕ on an example and refer the reader to [30] for the actual proof.

Consider

$$\phi = \exists x \cdot \forall y \cdot \exists z \cdot (x \wedge \neg y \wedge z) \vee (y \wedge \neg z)\ .$$

The hierarchical agent A_1 contains the following rules.

$$\begin{array}{llll} x' \leftarrow & \neg x' \leftarrow & y' \leftarrow & \neg y' \leftarrow \\ z' \leftarrow & \neg z' \leftarrow & sat' \leftarrow x', \neg y', z' & sat' \leftarrow y', \neg z' \end{array}$$

We have 8 possible agent answer sets for $A_1(\emptyset)$, i.e. $I_1 = \{x, y, z\}$, $I_2 = \{x, y, \neg z, sat\}$, $I_3 = \{x, \neg y, z, sat\}$, $I_4 = \{x, \neg y, \neg z\}$, $I_5 = \{\neg x, y, z\}$, $I_6 = \{\neg x, y, \neg z, sat\}$, $I_7 = \{\neg x, \neg y, z\}$ and $I_8 = \{\neg x, \neg y, \neg z\}$.

Clearly, for $1 \leq i \leq 8$, (I_i) is a hierarchical answer set of (A_1).

The second hierarchical agent A_2 is defined by $P_\exists^3$ and thus contains the following rules.

$$\begin{array}{llll} \mathbf{pass}(\{x, \neg x, y, \neg y\}) \leftarrow & z' \leftarrow & \neg z' \leftarrow & sat' \leftarrow x', \neg y', z' \\ \neg sat' \leftarrow & \leftarrow \neg sat' & \leftarrow sat & sat' \leftarrow y', \neg z' \end{array}$$

Now, feeding I_1 to A_2 yields I_2 as the single agent answer set, which is clearly inconsistent with I_1, yielding that (I_1, I_2) cannot be a hierarchical answer set of the

HAS (A_1, A_2). Further, for I_2 we have that $I_2 \in \mathcal{AG}(A_1)$, yielding that (I_2, I_2) is clearly consistent, which implies that (I_1, I_1) is neither a hierarchical answer set of (A_1, A_2).

On the other hand, $A_2(I_2)$ yields I_2 as the single agent answer set which is clearly consistent with itself, yielding that (I_2, I_2) is a hierarchical answer set of (A_1, A_2).

In case of the input I_7, the agent program $A_2(I_7)$ has no extended answer sets and I_7 is returned as the single agent answer set. As I_7 is consistent with itself, (I_7, I_7) is a hierarchical answer set of (A_1, A_2).

One can check in similar ways that $\mathcal{AG}((A_1, A_2))$ contains 5 interpretations, i.e. $\mathcal{AG}((A_1, A_2)) = \{(I_2, I_2), (I_3, I_3), (I_6, I_6), (I_7, I_7), (I_8, I_8)\}$. It is not difficult to see that for each of these hierarchical answer sets it holds that $\exists z \cdot (x \wedge \neg y \wedge z) \vee (y \wedge \neg z)$ when x and y are taken as in the interpretation iff the literal sat is contained in the hierarchical answer set.

The third and final hierarchical agent A_3 is given by $P^2_\forall$ and thus contains the following rules.

$$
\begin{array}{llll}
 & \textbf{pass}(\{x, \neg x\}) \leftarrow & & \leftarrow sat' \\
y' \leftarrow & \neg y' \leftarrow & \neg sat \leftarrow & sat' \leftarrow x', \neg y', z' \\
z' \leftarrow & \neg z' \leftarrow & \leftarrow \neg sat & sat' \leftarrow y', \neg z'
\end{array}
$$

When providing A_3 with the input I_2, we have $\mathcal{AS}(A_3, I_2) = \{I_1, I_4\}$, none of them consistent with I_2. On the other hand, neither for I_1 nor I_4 there is a $T \in \mathcal{AG}((A_1, A_2))$ such that I_1 or I_4 is consistent with T, yielding that (I_2, I_2, I_2) is a hierarchical answer set of (A_1, A_2, A_3) in this case, i.e. A_3 passes the input I_2 as a result as it cannot disprove $\forall y \cdot \exists z \cdot (x \wedge \neg y \wedge z) \vee (y \wedge \neg z)$ for the chosen truth value of x in I_2. In a similar way one can check that also $(I_3, I_3, I_3) \in \mathcal{AG}((A_1, A_2, A_3))$.

On the other hand, when feeding A_3 with I_6, we get $\mathcal{AS}(A_3, I_6) = \{I_5, I_7, I_8\}$, again none of them consistent with the input I_6. However, this time, such as for I_7, there exists an interpretation $T \in \mathcal{AG}((A_1, A_2))$ such that T is consistent with I_7, i.e. $T = (I_7, I_7)$, and as a result (I_6, I_6, I_6) is rejected as a hierarchical answer set for (A_1, A_2, A_3).

When the agent A_3 is given either I_7 or I_8, it cannot produce any extended answer sets and thus returns I_7 or I_8 respectively as the single agent answer set, yielding that (I_7, I_7, I_7) and (I_8, I_8, I_8) are both hierarchical answer sets for (A_1, A_2, A_3).

This time, one can check that for each hierarchical answer set in $\mathcal{AG}((A_1, A_2, A_3))$ it holds that $\forall y \cdot \exists z \cdot (x \wedge \neg y \wedge z) \vee (y \wedge \neg z)$ for x taken as in the interpretation iff the literal sat is contained in the hierarchical answer set. From this it follows that ϕ is valid iff there exists a hierarchical answer set $I \in \mathcal{AG}((A_1, A_2, A_3))$ such that the literal sat is contained in the interpretation. In our example, I_2 is such a hierarchical answer set and one can check that ϕ holds when assuming x is true. $\square$

5 Relationships to Other Approaches

In [3], answer set optimization (ASO) programs are presented. Such ASO programs consist of a generator program and a sequence of optimizing programs. To perform the optimization, the latter programs use rules of the form $c_1 < \cdots < c_n \leftarrow \beta$ which intuitively read: when β is true, making c_1 true is the most preferred option and only when c_1 cannot be made true, the next best option is to make c_2 true, ... Solutions of the generator program that are optimal w.r.t. the first optimizing program and, among those, are optimal w.r.t. the second optimizing program, and so on, are called preferred solutions for the ASO program.

The framework of ASO programming can be simply adapted to the setting of agents, i.e. just consider the generator program as agent A_1 and the optimizing programs as agents $A_2, \ldots, A_n$. The resulting semantics is very similar to our approach. However, ASO programs are far more limited w.r.t. their expressiveness. It turns out that the expressiveness of an ASO program does not depend on the length of the sequence of optimizing programs, but it is always Σ_2^P-complete. This yields that ASO programs can easily be captured by the presented agent systems in this paper using two single agents. How these two agents simulating ASO programs can be constructed is subject to further research.

Weak constraints were introduced in [5] as a relaxation of the concept of a constraint. Intuitively, a weak constraint is allowed to be violated, but only as a last resort, meaning that one tries to minimize the number of violated constraints. Additionally, weak constraints are allowed to be hierarchically layered by means of a sequence of sets of weak constraints. Intuitively, one first chooses the answer sets that minimize the number of violated constraints in the first set of weak constraints in the sequence, and then, among those, one chooses the answer sets that minimize the number of violated constraints in the second set, etc.

Again, this approach can be "agentized" in a straightforward manner and will look similar to our approach. This time the complexity of such a system, independent of the number of sets of weak constraints, is at most Δ_3^P-complete. Thus, using the presented agent system from Section 3 with three single agents will suffice to capture the most expressive form of that formalism.

In [19, 31], hierarchies of preferences on a single program are presented. The preferences are expressible on both the literals and the rules in that program. It is shown that for a sequence of n agents the complexity of the system is Σ_{n+1}^P-complete. The semantics proposed in Section 3 is a generalization of that approach: instead of using one global program with agents only using preferences on that program, we equip each agent with her own, in general different, program and let her implement whatever optimizing strategy she wants. To capture a hierarchy of n preference relations, we need $n+1$ optimizing agents: the first one will correspond with the global program, while the rest will correspond to the n preference relations. The system described in Example 7 can be seen as a translation of such a preference hierarchy. Intuitively, agent A_2 describes the preference relation[6] $experienced < inexperienced$, while A_3 implements the relation $young < old$; $inexperienced < experienced$. Finally, A_4 corresponds to the single preference $female < male$.

In [25] a theory for coordinating agents is presented. When two agents A and B, which are represented by extended disjunctive programs, coordinate their answer sets, they can either opt for generating the union of their answer sets or the intersection. Our

[6] The expression $a < b$ means a is preferred upon b.

work shows some resemblance to the latter, yet our agents are in a hierarchy giving more power to agents higher up. Furthermore, our method is universal.

[8, 10] also present a multi-agent framework, LAIMA, were the agents are represented as a logic program. LAIMA allows agents to be connected in any sort of way, including loops. The HAS system presented in this paper places the most influencial agents at the start of the sequence of agents, providing a top-down approach. The LAIMA system does exactly the opposite, it provides a bottom-up approach by starting the reasoning at the bottom of the hierachy and moving up. The major differences with our approach is that LAIMA does not deal with agents failing to generate an answer set and that they allow agents to contradict each other.

In the Minerva architecture [21], the authors build their agents out of subagents that work on a common knowledge base written as a MDLP (Multi-Dimensional Logic Program) which is an extension of Dynamic Logic Programming. It can be shown that MDLP can be translated to extended logic programs such that their stable models match our answer sets.

6 Conclusions and Directions for Further Research

We presented a framework suitable for solving hierarchical decision problems using simple logic programming agents that cooperate via a sequential communication channels. The resulting semantics turns out to be very expressive, as it essentially covers the polynomial hierarchy, thus enabling further complex applications. The framework could be used to develop implementations for diagnostic systems at the third level of the polynomial hierarchy [11, 13, 34].

Future work comprises the development of a dedicated implementation of the approach, using existing answer set solvers, e.g. DLV [16] or SMODELS [26], possibly in a distributed environment. Such an implementation will use a control structure that communicates candidate solutions between consecutive agents. For the implementation of this control loop and the communication between the agents, we foresee the use of JADE [20] and Protégé [24] in much the same way as it is been done for the LAIMA system [8].

In the context of an implementation, it is also interesting to investigate which conditions an agent has to fulfil in order for it not to lift the complexity up one level in the polynomial hierarchy, yielding possible optimizations of the computation and communication process.

Once the system is implemented we will have the opportunity to work on larger applications. One of our goals, is to try to incorporate the ALIAS [6] system, an agent architecture for legal reasoning based on abductive logic, into ours. The Carell multi-agent [36] for allocation organs and tissue would be an interesting test case.

In terms of integration it would be nice to see how HAS could possibly work together with agents written for the Dali [7] or Socs [15, 27] platforms, two agent platforms using logic programming languages to model the agents.

Present, we only work with a linear sequence of communication channels. We plan to look into a broader class of communication structures, like for example trees or more generally, a (strict) partial ordering of agents.

Finally, we would like to experiment with the language(s) used for our agents. The definition of hierarchical agent system and the corresponding hierarchical answer set does not necessary have to rely on the representation language of the agents, as long as they can produce agent answer sets.

References

[1] C. Baral and M. Gelfond. Reasoning agents in dynamic domains. In *Logic-based artificial intelligence*, pages 257–279. Kluwer Academic Publishers, 2000.

[2] G. Brewka, I. Niemela, and T. Syrjanen. Implementing ordered disjunction using answer set solvers for normal programs. In Flesca et al. [17], pages 444–455.

[3] G. Brewka, I. Niemelä, and M. Truszczynski. Answer set optimization. In G. Gottlob and T. Walsh, editors, *IJCAI*, pages 867–872. Morgan Kaufmann, 2003.

[4] F. Buccafurri and G. Gottlob. Multiagent compromises, joint fixpoints, and stable models. In A. C. Kakas and F. Sadri, editors, *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part I*, volume 2407 of *Lecture Notes in Computer Science*, pages 561–585. Springer, 2002.

[5] F. Buccafurri, N. Leone, and P. Rullo. Strong and weak constraints in disjunctive datalog. In *Proceedings of the 4th International Conference on Logic Programming (LPNMR '97)*, pages 2–17, 1997.

[6] A. Ciampolini and P. Torroni. Using abductive logic agents for modeling the judicial evaluation of crimimal evidence. *Applied Artificial Intelligence*, 18:251–275, 2004.

[7] S. Costantini and A. Tocchio. Context-based Commmonsense Reasoning in the DALI Logic Programmming Language. In T. Blackburn, C. Ghidini, R. M. Turner, and G. F., editors, *Proceedings of the 4th International and Interdisciplinary Conference, Context 2003*, volume 2680 of *Lecture Notes in Artificial Intelligence*, Stanford, CA, June 2003. Springer-Verlag, Berlin.

[8] M. De Vos, T. Crick, J. Padget, M. Brain, O. Cliffe, and J. Needham. A Multi-agent Platform using Ordered Choice Logic Programming. In *Declarative Agent Languages and Technologies (DALT'05)*, July 2005. http://www.doc.ic.ac.uk/ ue/DALT-2005/vos.shtml.

[9] M. De Vos and D. Vermeir. Choice Logic Programs and Nash Equilibria in Strategic Games. In J. Flum and M. Rodríguez-Artalejo, editors, *Computer Science Logic (CSL'99)*, volume 1683 of *LNCS*, pages 266–276, Madrid, Spain, 1999. Springer Verslag.

[10] M. De Vos and D. Vermeir. Extending Answer Sets for Logic Programming Agents. *Annals of Mathematics and Artifical Intelligence*, 42(1–3):103–139, Sept. 2004. Special Issue on Computational Logic in Multi-Agent Systems.

[11] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. The diagnosis frontend of the dlv system. *AI Communications*, 12(1-2):99–111, 1999.

[12] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. The DLV^k planning system. In Flesca et al. [17], pages 541–544.

[13] T. Eiter and G. Gottlob. The complexity of logic-based abduction. *Journal of the Association for Computing Machinery*, 42(1):3–42, 1995.

[14] T. Eiter and A. Polleres. Towards automated integration of guess and check programs in answer set programming. In V. Lifschitz and I. Niemelä, editors, *LPNMR*, volume 2923 of *LNCS*, pages 100–113, Fort Lauderdale, FL, USA, January 2004. Springer.

[15] U. Endriss, N. Maudet, F. Sadri, and F. Toni. Logic-based agent communication protocols. In F. Dignum, editor, *Advances in Agent Communication*, volume 2922 of *LNAI*, pages 91–107. Springer-Verlag, 2004.

[16] W. Faber and G. Pfeifer. dlv homepage. http://www.dbai.tuwien.ac.at/proj/dlv/.

[17] S. Flesca, S. Greco, N. Leone, and G. Ianni, editors. *European Conference on Logics in Artificial Intelligence (JELIA '02)*, volume 2424 of *Lecture Notes in Artificial Intelligence*, Cosenza, Italy, September 2002. Springer Verlag.
[18] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. A. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium*, pages 1070–1080, Seattle, Washington, August 1988. The MIT Press.
[19] S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Hierarchical decision making by autonomous agents. In J. J. Alferes and J. A. Leite, editors, *Logics in Artificial Intelligence, 9th European Conference, Proceedings, JELIA 2004*, volume 3229 of *LNCS*, pages 44–56. Springer, 2004.
[20] Jade:. `http://jade.tilab.com/`.
[21] J. A. Leite, J. J. Alferes, and L. M. Pereira. Minerva - a dynamic logic programming agent architecture. In J. J. Meyer and M. Tambe, editors, *Intelligent Agents VIII*, number 2002 in LNAI, pages 141–157. Springer-Verlag, 2002.
[22] V. Lifschitz. Answer set programming and plan generation. *Journal of Artificial Intelligence*, 138(1-2):39–54, 2002.
[23] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.
[24] Protégé. `http://protege.stanford.edu/`.
[25] C. Sakama and K. Inoue. Coordination between logical agents. In *Proceedings of the 5th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA-V)*, volume 3487 of *LNAI*, pages 161–177. Springer, 2005.
[26] P. Simons. smodels homepage. http://www.tcs.hut.fi/Software/smodels/.
[27] SOCS:. `http://lia.deis.unibo.it/research/socs/`.
[28] T. Soininen and I. Niemelä. Developing a declarative rule language for applications in product configuration. In *Proceedings of the First International Workshop on Practical Aspects of Declarative Languages (PADL '99)*, LNCS, San Antonio, Texas, 1999. Springer.
[29] L. Stockmeyer and A. Meyer. Word problems requiring exponential time. In *Proceedings of the 5th ACM Symposium on Theory of Computing (STOC '73)*, pages 1–9, 1973.
[30] D. Van Nieuwenborgh, M. De Vos, S. Heymans, and D. Vermeir. Hierarchical knowledge aquisition in multi-agent systems using answer set programming. Technical report, Vrije Universiteit Brussel, Dept. of Computer Science, 2005.
[31] D. Van Nieuwenborgh, S. Heymans, and D. Vermeir. On programs with linearly ordered multiple preferences. In B. Demoen and V. Lifschitz, editors, *Proceedings of 20th International Conference on Logic Programming (ICLP 2004)*, number 3132 in LNCS, pages 180–194. Springer, 2004.
[32] D. Van Nieuwenborgh and D. Vermeir. Preferred answer sets for ordered logic programs. In Flesca et al. [17], pages 432–443.
[33] D. Van Nieuwenborgh and D. Vermeir. Order and negation as failure. In C. Palamidessi, editor, *Proceedings of the 19th International Conference on Logic Programming*, volume 2916 of *LNCS*, pages 194–208. Springer, 2003.
[34] D. Van Nieuwenborgh and D. Vermeir. Ordered diagnosis. In Vardi and Voronkov [35], pages 244–258.
[35] M. Y. Vardi and A. Voronkov, editors. *Proceedings of the 10th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR2003)*, volume 2850 of *Lecture Notes in Artificial Intelligence*, Almaty, Kazachstan, 2003. Springer Verlag.
[36] J. Vázquez-Salceda, J. Padget, U. Cortés, A. López-Navidad, and F. Caballero. Formalizing an electronic institution for the distribution of human tissues. *Artificial Intelligence in Medicine*, 27(3):233–258, 2003. published by Elsevier.
[37] T. Wakaki, K. Inoue, C. Sakama, and K. Nitta. Computing preferred answer sets in answer set programming. In Vardi and Voronkov [35], pages 259–273.