Preferential Reasoning on a Web of Trust

Stijn Heymans, Davy Van Nieuwenborgh[⋆], and Dirk Vermeir[⋆⋆]

Dept. of Computer Science
Vrije Universiteit Brussel, VUB
Pleinlaan 2, B1050 Brussels, Belgium
{sheymans,dvnieuwe,dvermeir}@vub.ac.be

Abstract. We introduce a framework, based on logic programming, for preferential reasoning with agents on the Semantic Web. Initially, we encode the knowledge of an agent as a logic program equipped with call literals. Such call literals enable the agent to pose yes/no queries to arbitrary knowledge sources on the Semantic Web, without conditions on, e.g., the representation language of those sources. As conflicts may arise from reasoning with different knowledge sources, we use the extended answer set semantics, which can provide different strategies for solving those conflicts. Allowing, in addition, for an agent to express its preference for the satisfaction of certain rules over others, we can then induce a preference order on those strategies. However, since it is natural for an agent to believe its own knowledge (encoded in the program) but consider some sources more reliable than others, it can alternatively express preferences on call literals. Finally, we show how an agent can learn preferences on call literals if it is part of a web of trusted agents.

1 Introduction

The current WWW is a gigantic pool of data, where one can easily imagine two web sites saying the opposite. Human users are capable of deciding which sources they find trustworthy or not (irrespective of the fact whether they actually are or not). Semantic Web software agents [18] on the other hand would have an equally vast amount of data at their disposition, but a far more difficult time differentiating between good and bad information.

In this paper, we will gradually build a (abstract) software agent, i.e. an entity on a web of trust that can reason with a diverse pool of (possibly mutually inconsistent) knowledge sources. The basic underlying reasoning framework we use for such an agent is *answer set programming (ASP)* [13, 3], a logic programming paradigm with a stable model semantics for negation as failure. A *logic program* corresponds to knowledge one wishes to represent, or, more specifically, to an encoding of a particular problem, e.g. a planning problem [24, 9]; the *answer sets* of the program then provide its intentional knowledge, or the solutions of the encoded problem, e.g. a plan for a planning problem.

[⋆] Supported by the FWO.

[⋆⋆] This work was partially funded by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2001-37004 WASP project.

A traditional logic program has a limited view on the world; it restricts itself to its own knowledge and does not allow calls to external sources. In a first phase, to construct suitable Semantic Web agents, we thus introduce *call literals* in rules, e.g., a rule $tr_1 : \neg train \leftarrow geo1.300km(brussels, madrid)$, where $geo1.300km(brussels, madrid)$ is a call literal and $\neg train$ a normal literal. The rule reads "if according to the $geo1$ ontology Brussels is more than 300 km away from Madrid, one should not go by train". The word "ontology" is slightly misleading, since $geo1$ can be anything: an OWL DL [4] knowledge base, an SQL database, RDF data, another agent, anything. In order to establish a suitable semantics for such call literals, we associate with each call literal in a program an instance of a decision problem, e.g., satisfiability checking in OWL DL, checking whether a tuple is in the database, ... An evaluation function then assigns **true** or **false** to the call literal, depending on the corresponding instance. Technically, programs with calls are a subclass of logic programs with generalized quantifiers Q_C [10], where a generalized quantifier Q_C checks whether a relation defined by the program is in a class of structures C. In the proposed setting, every call literal corresponds to a class of structures C that is a singleton set containing some literal if the instance of the decision problem associated with that call literal returns **true**.

In contrast with approaches as in [5, 17, 25, 23, 19] where one attempts to reduce reasoning in description logics (DLs) [2] to logic programming or approaches biased more towards the integration of description logics and logic programming reasoning [8, 29, 11], the proposed framework does not restrict itself to DLs, knowledge can be represented in any language with associated reasoning procedures; agents that want to use the knowledge only have to know how to call those procedures.

Besides making calls to sources, agents have to be able to cope with conflicts, e.g., add to the above train rule that if Brussels and Madrid are not divided by water, one should take the train: $tr_2 : train \leftarrow not\ geo2.dividedwater(brussels, madrid)$. If the call to $geo1$ returns **true**, claiming that Brussels is indeed more than 300 km away from Madrid, and that the call to $geo2$ returns **false** (and is thus faulty), this leads to a conflict since tr_1 deduces $\neg train$ and tr_2 deduces $train$. The normal answer set semantics has no answer sets for this program, which is not feasible on the Semantic Web – we do not want an agent to stay indecisive on different contradicting sources. The extended answer set semantics and its notion of *defeat* loosens up the normal answer set semantics by allowing rules to remain unsatisfied provided there is a competing rule (i.e. a rule with opposite head) that is applied (both the head and body are true). The above program results then in the two *extended answer sets* $\{train\}$ and $\{\neg train\}$, representing the possible alternatives for the conflict, where tr_1, respectively tr_2, is defeated.

The agent can then choose among those possible solutions based on a preference on the satisfaction of rules, e.g., $tr_1 < tr_2$, indicates that the agent prefers to satisfy tr_1 over tr_2. This preference naturally induces an order on its extended answer sets:$\{\neg train\} \sqsubseteq \{train\}$. A wide variety of applications of agents with preferences are imaginable, e.g. to guide service discovery on mobile devices [31].

In the context of the Semantic Web, a preference on call literals seems more natural than an order on the agent's own rules: an agent generally assumes its own rules are correct, whereas the uncertain part, and hence the part that may introduce conflicts, are the external calls. Based on criteria such as authority or reliability the agent can then express its preference for certain calls. Furthermore, we show a translation of an order on call literals to an order on the rules of the agents.

What if the agent does not know which calls are more reliable than others; can it still make an educated guess regarding its preferences? The *web of trust* [14, 16, 15, 28, 7] provides an architecture on which preferential reasoning for agents without (or with incomplete) preferences can be realized.

In [28], a web of trust is essentially a graph of agents where edges have a weight in $[0, 1]$, indicating the amount of trust an agent has in its direct neighbors. Moreover, every user can have a belief, a number in $[0, 1]$, in logical statements. The *merged belief* in a logical statement, i.e. taking into account the beliefs in that statement of trusted agents, can be computed in a large number of ways, e.g., one can demand that the amount of trust between users is at most the minimal trust weight on a path between them or that the further away an agent is, the lower the trust in that agent should be [16]. In the TRELLIS system [14] users rate information sources and, assuming different users rate common sources, TRELLIS rates sources averaging over the ratings of different users.

Relating this to our approach, the beliefs in statements correspond to preferences on call literals. Furthermore, in order to construct agents on top of any web of trust, we do not presuppose any conditions on the trust metric, i.e. the method to calculate the merged trust given a web of trust, but one: it must be possible to associate with every agent a sequence of trusted agents ordered according to trustworthiness. Given, for each agent, such an ordered sequence, we then complete the preferences of an agent by considering its own preferences and adding further preferences according to its trusted agents.

The remainder of the paper is organized as follows. In Section 2, we define the preferred answer set semantics. Section 3 extends the preferred answer set semantics with the possibility to define call literals and their accompanying calls. In Section 4, we define a preference order on literals and a method for constructing this order based on a web of trust. Finally, Section 5 contains conclusions and directions for further research. Due to space restrictions, proofs have been omitted but can be found in [20].

2 Preliminaries: Preferred Answer Set Programming

We introduce the extended answer set semantics as in [30]. A *literal* is an atom a or a classically negated atom $\neg a$; an *extended literal* is a literal l or a literal preceded with the *negation as failure* symbol *not*: $not\ l$. A *program* is a finite set of rules $\alpha \leftarrow \beta$ where α, the *head*, is a set of literals with $|\alpha| \leq 1$, i.e. α is empty or a singleton, and β, the *body*, is a finite set of extended literals. We usually denote a rule as $a \leftarrow \beta$ or $\leftarrow \beta$, and we call the latter a *constraint*. The positive part of the body is $\beta^+ = \{l \mid l \in \beta, l \text{ literal}\}$, the negative part is $\beta^- = \{l \mid not\ l \in \beta\}$, e.g. for $\beta = \{a, not\ \neg b, not\ c\}$, we have that $\beta^+ = \{a\}$ and $\beta^- = \{\neg b, c\}$. For a set of literals α, $not\ \alpha = \{not\ a \mid a \in \alpha\}$, and $\alpha^* = \alpha \cup not\ \alpha$.

The *Herbrand Base* $\mathcal{B}_P$ of a program P is the set of all atoms that can be formed using the language of P. Let $\mathcal{L}_P$ be the set of literals that can be formed with P, i.e. $\mathcal{L}_P = \mathcal{B}_P \cup \neg\mathcal{B}_P$. For a set X of literals, we take $\neg X = \{\neg l \mid l \in X\}$ where $\neg\neg a$ is a; X is *consistent* if $X \cap \neg X = \emptyset$. An *interpretation* I of P is any consistent subset of

$\mathcal{L}_P$. For a literal l, we write $I \models l$, if $l \in I$, which extends for extended literals $not\ l$ to $I \models not\ l$ if $I \not\models l$. In general, for a set of extended literals X, $I \models X$ if $I \models x$ for every extended literal $x \in X$. A rule $r : a \leftarrow \beta$ is *satisfied* w.r.t. I, denoted $I \models r$, if $I \models a$ whenever $I \models \beta$, i.e. r is *applied* whenever it is *applicable*. A constraint $\leftarrow \beta$ is satisfied w.r.t. I if $I \not\models \beta$. The set of satisfied rules in P w.r.t. I is the *reduct* P_I.

For a program P without negation as failure, an interpretation I is a *model* of P if I satisfies every rule in P, i.e. $P_I = P$; it is an *answer set* of P if it is a minimal model of P, i.e. there is no model J of P such that $J \subset I$. For programs P containing *not*, the *GL-reduct* w.r.t. an interpretation I is P^I, where P^I contains $\alpha \leftarrow \beta^+$ for $\alpha \leftarrow \beta$ in P and $\beta^- \cap I = \emptyset$. I is an *answer set* of P if I is an answer set of P^I. A rule $a \leftarrow \beta$ is *defeated* w.r.t. I if there is a *competing* rule $\neg a \leftarrow \gamma$ that is applied w.r.t. I, i.e. $\{\neg a\} \cup \gamma \subseteq I$. An *extended answer set* I of a program P is an answer set of P_I such that all rules in $P \backslash P_I$ are *defeated*.

Consider a program P indicating that one wants to take the train (t_1), that if the distance to the destination is more than 300 km, one does not want to take the train (t_2), and that the distance is actually more than 300 km (t_3).

$$\begin{array}{ll} t_1 : \quad train \leftarrow & \qquad t_2 : \neg train \leftarrow 300km \\ t_3 : 300km \leftarrow & \end{array}$$

This program no answer sets and two extended answer sets $M_1 = \{300km, train\}$ and $M_2 = \{300km, \neg train\}$: there is no competing rule for t_3 such that it must be satisfied and every extended answer set must contain $300km$. The rule t_2 is not satisfied in M_1 (the body is true while the head is not), but it is defeated since the competing rule t_1 is applied in M_1. In M_2, t_1 is defeated by the applied t_2.

Resolving conflicts by defeating rules thus leads to different alternative extended answer sets. Usually however, a user may have some particular preferences on the satisfaction of the rules. As in [30], we impose a strict partial order[1] $<$ on the rules in P, indicating these preferences, which results in an *ordered logic program (OLP)* $\langle P, < \rangle$.This preferential ordering will induce an ordering $\sqsubseteq$ among the possible alternative extended answer sets as follows: for interpretations M and N of P, M is "more preferred" than N, denoted $M \sqsubseteq N$, if $\forall r_2 \in P_N \backslash P_M \cdot \exists r_1 \in P_M \backslash P_N \cdot r_1 < r_2$. Intuitively, for every rule that is satisfied by N and not by M, and which thus appears to be a counterexample for M being better than N, there is a better rule that is satisfied by M and not by N, i.e. M can counter the counterexample of N. We have that M is "strictly better" than N, $M \sqsubset N$, if $M \sqsubseteq N$ and not $N \sqsubseteq M$. An extended answer set is a *preferred answer set* of $\langle P, < \rangle$ if it is minimal w.r.t. $\sqsubset$ among the extended answer sets.

Considering the extended answer sets for the train example, we have that $P_{M_1} = \{t_1, t_3\}$ and $P_{M_2} = \{t_2, t_3\}$. If we prefer going by train over not going by train, i.e. $t_1 < t_2$, we have that $M_1 \sqsubseteq M_2$ since for every rule in $P_{M_2} \backslash P_{M_1} = \{t_2\}$, there is a better one in $P_{M_2} \backslash P_{M_1} = \{t_1\}$. Since $M_2 \not\sqsubseteq M_1$, we have that $M_1 \sqsubset M_2$, making M_1 the only preferred answer set of the program.

For reference later on in the paper, we briefly restate the complexity results from [30] for the preferred answer set semantics. Checking whether a program has an extended answer set containing a particular literal is NP-complete, while checking whether

[1] A strict partial order on X is an anti-reflexive and transitive relation on X.

an ordered program has a preferred answer set containing a particular literal is Σ_2^P-complete. Recall that NP represents the problems that are nondeterministically decidable in polynomial time, while Σ_2^P is NP^{NP}, i.e. the problems that are nondeterministically decidable in polynomial time using an NP oracle, where an NP oracle is a subroutine capable of solving NP problems in unit time. For an arbitrary complexity class C, the class P^C represents those problems that are deterministically decidable in polynomial time with an oracle for problems in C. Finally, we mention the complexity class EXPTIME (NEXPTIME) of problems deterministically (nondeterministically) decidable in exponential time. A language L is called complete for a complexity class C if both L is in C and L is hard for C. Showing that L is hard is normally done by reducing a known complete decision problem to a decision problem in L. More on complexity in general can be found in, e.g., [27].

3 Preferred Answer Set Programming with Calls

We extend preferred answer set programming with call literals. Take, for example, a program with facts declaring kine to be a movie theater, pizzi and ilpast restaurants, and times 8 P.M. and 10 P.M.

$$\begin{array}{rr} \mathit{movies}(\mathit{kine}) \leftarrow & \mathit{time}(\mathit{8pm}) \leftarrow \\ \mathit{rest}(\mathit{pizzi}) \leftarrow & \mathit{time}(\mathit{10pm}) \leftarrow \\ \mathit{rest}(\mathit{ilpast}) \leftarrow & \end{array}$$

We have a rule p that produces a plan for a night out to a restaurant Rest and a movie theater Movies at respective times $\mathit{Time1}$ and $\mathit{Time2}$.

$$\begin{array}{l} p : \mathit{plan}(\mathit{Rest} : \mathit{rest}, Time1 : \mathit{time}, Movies : \mathit{movies}, Time2 : \mathit{time}) \leftarrow \\ \qquad \mathit{Rest}.\mathit{res}(Time1), \mathit{geo}.\mathit{near}(\mathit{Rest}, Movies), Time1 \neq Time2, \\ \qquad\qquad\qquad\qquad \mathit{not}\ \mathit{otherpl}(\mathit{Rest}, Time1, Movies, Time2) \end{array}$$

The *call literal* $\mathit{Rest}.\mathit{res}(\mathit{Time1})$ represents a query to a restaurant's knowledge to check whether one can reserve at a time. The call literal, $\mathit{geo}.\mathit{near}(\mathit{Rest}, \mathit{Movies})$, queries some knowledge source geo in order to ensure that the restaurant and the movie theater are located in each other's vicinity. The inequality $Time1 \neq Time2$ expresses that dinner time must be different from the movie's time. We used syntactic sugar for typing arguments, e.g. $\mathit{Rest} : \mathit{rest}$ indicates that the variable Rest is of type rest. Formally, we define a rule with typing $p(T : t, \boldsymbol{x}) \leftarrow \beta$ as the rule $p(T, \boldsymbol{x}) \leftarrow t(T), \beta$. The extended literal $\mathit{not}\ \mathit{otherpl}(\mathit{Rest}, \mathit{Time1}, \mathit{Movies}, \mathit{Time2})$ ensures that there is only one plan in each result: $o_1 : \mathit{otherpl}(\mathit{Resta}, Time1a, Moviesa, Time2a) \leftarrow$ $\mathit{plan}(\mathit{Restb}, Time1b, Moviesb, Time2b), \mathit{Resta} \neq \mathit{Restb}$, and similar o_2, o_3, and o_4, with inequalities on the Time and Movies variables.

Furthermore, we want a classification of theaters that screen romantic movies. We query two repositories that are able to verify whether a movie theater has romantic movies programmed: $moviedb1.roman(Movies)$ and $moviedb2.roman(Movies)$.

$$\begin{aligned}
r_1 &: \quad roman(Movies : movies) \leftarrow moviedb1.roman(Movies)\\
r_2 &: \neg roman(Movies : movies) \leftarrow not\ moviedb1.roman(Movies)\\
r_3 &: \quad roman(Movies : movies) \leftarrow moviedb2.roman(Movies)\\
r_4 &: \neg roman(Movies : movies) \leftarrow not\ moviedb2.roman(Movies)
\end{aligned}$$

Finally, the night out might be a date or not (rule d, where a rule of the form $a \vee \neg a \leftarrow$ is shorthand for the rules $a \leftarrow not\ \neg a$ and $\neg a \leftarrow not\ a$), and we have a constraint indicating that a plan for a date should involve a movie theater where romantic movies are screened:

$$\begin{aligned}
d &: date \vee \neg date \leftarrow\\
c &: \qquad\qquad \leftarrow plan(Rest, Time1, Movies, Time2), date, \neg roman(Movies)
\end{aligned}$$

In the following, we assume, as is usual in logic programming, that programs are *grounded*: each variable is replaced by all possible constants. In the presence of call literals, we further generalize this such that every word starting with a capital letter is replaced by all possible constants. The rule p thus yields, among others,

$$\begin{aligned}
&plan(pizzi, 8pm, kine, 10pm) \leftarrow\\
&\quad pizzi.res(8pm), geo.near(pizzi, kine), not\ otherpl(pizzi, 8pm, kine, 10pm)
\end{aligned}$$

We grounded the words $Rest$ and $Time1$ in $Rest.res(Time1)$ by $pizzi$ and $8pm$ respectively. Additionally, grounding takes into account inequalities and subsequently removes them from the rules: $Time1$ and $Time2$ are grounded by different constants. Grounding does not care for semantics, e.g., the literal $8pm.res(kine)$ is a valid, albeit nonsensical, grounding for $Rest.res(Time1)$.

Syntactically, a ground program with calls does not differ from a ground program without calls: a literal is only a call literal if it is explicitly associated with a particular instance of a decision problem.

Definition 1. *A* call semantics *for a program R is a mapping $\sigma : \mathcal{C}_R \subseteq \mathcal{L}_R \rightarrow Inst$ from a designated set of* call literals *$\mathcal{C}_R$ in R to instances $Inst$ of decision problems D.*

We relate every instance in $Inst$ to its decision problem by a mapping $d : Inst \rightarrow D$ such that $d(Inst) = D$. A call semantics is *well-defined* if every decision problem $d \in D$ is decidable and has an associated complexity $\mathsf{comp}(d)$. The *call complexity* $\mathsf{comp}(\sigma)$ of a well-defined σ is the complexity class $\bigcup\{\mathsf{comp}(d) \mid d \in D\}$. For the grounding R of the above program, we define the call literals $\mathcal{C}_R = \{pizzi.res(8pm)$, $pizzi.res(10pm), ilpast.res(8pm), ilpast.res(10pm), geo.near(pizzi, kine)$, $geo.near(ilpast, kine), moviedb1.roman(kine), moviedb2.roman(kine)\}$, with σ as in Table 1. Thus, e.g., $\sigma(pizzi.res(8pm))$ is an instance of *instance checking* for OWL DL ontologies, $\sigma(moviedb1.roman(kine))$ is an instance of the problem that involves checking whether there is an answer set of a program containing a certain literal, and, $\sigma(geo.near(pizzi, kine))$ is some other unspecified instance of a decidable problem. Assuming the complexity of the latter is polynomial, we have, with the NEXPTIME complexity for instance checking in OWL DL [22] and NP complexity for the answer

Table 1. Call Semantics σ

$\sigma(\mathit{pizzi}.\mathit{res}(\mathit{8pm}))$ =	'is $\mathit{res}(\mathit{8pm})$ in model of OWL DL ontology pizzi'
$\sigma(\mathit{pizzi}.\mathit{res}(\mathit{10pm}))$ =	'is $\mathit{res}(\mathit{10pm})$ in model of OWL DL ontology pizzi'
$\sigma(\mathit{ilpast}.\mathit{res}(\mathit{8pm}))$ =	'is $\mathit{res}(\mathit{8pm})$ in model of OWL DL ontology ilpast'
$\sigma(\mathit{ilpast}.\mathit{res}(\mathit{10pm}))$ =	'is $\mathit{res}(\mathit{10pm})$ in model of OWL DL ontology ilpast'
$\sigma(\mathit{geo}.\mathit{near}(\mathit{pizzi}, \mathit{kine}))$=	'is pizzi near kine according to geo DB'
$\sigma(\mathit{geo}.\mathit{near}(\mathit{ilpast}, \mathit{kine}))$=	'is ilpast near kine according to geo DB'
$\sigma(\mathit{moviedb1}.\mathit{roman}(\mathit{kine}))$ =	'exists answer set of $\mathit{moviedb1}$ containing $\mathit{roman}(\mathit{kine})$'
$\sigma(\mathit{moviedb2}.\mathit{roman}(\mathit{kine}))$ =	'exists answer set of $\mathit{moviedb2}$ containing $\mathit{roman}(\mathit{kine})$'

set programming problem [6], that $\mathsf{comp}(\sigma) = \text{NEXPTIME} \cup \text{NP} \cup \text{P} = \text{NEXPTIME}$. The particular dot notation ($\mathit{Rest}.\mathit{res}(\mathit{Time})$) has thus no particular meaning in itself, apart from hinting that it might be a call of res to the object Rest. The identification of call literals and their semantics is the responsibility of the call semantics only.

In the following, we assume all call semantics are well-defined, and thus have an associated call complexity. Evaluating call literals amounts to evaluating the corresponding instance of the decision problem.

Definition 2. *Let σ be a call semantics for a program R. The* evaluation *of σ is a mapping* $\mathsf{eval}_\sigma : \mathcal{C}_R \cup \mathit{not}\ \mathcal{C}_R \rightarrow \{\textbf{true}, \textbf{false}\}$ *such that, for a call literal l,* $\mathsf{eval}_\sigma(l) =$ ***true*** *if $\sigma(l)$ evaluates to true and* $\mathsf{eval}_\sigma(l) =$ ***false*** *if $\sigma(l)$ evaluates to false. For a $\mathit{not}\ l \in \mathit{not}\ \mathcal{C}_R$, we define* $\mathsf{eval}_\sigma(\mathit{not}\ l) = \neg\mathsf{eval}_\sigma(l)$*. For a set of extended call literals X,* $\mathsf{eval}_\sigma(X) = \{\mathsf{eval}_\sigma(l) \mid l \in X\}$.

Definition 3. *A* program with calls (LPC) *is a pair $P = \langle R, \sigma\rangle$ where R is a program and σ is a call semantics for R.*

The semantics of LPCs is defined by a reduction to the extended answer set semantics for programs without calls. For a LPC $\langle R, \sigma\rangle$, we evaluate all call literals in R by means of σ. Since all call literals are interpreted as instances of decidable decision problems, such an evaluation returns either true or false for each call literal. Similar to the GL-reduct, the *call-free reduct* is then the original program R with call literals removed according to their evaluation: a call literal in the body that evaluates to false amounts to the removal of the rule since the rule can never contribute to an answer set; if a call literal in the body evaluates to true, one just removes it from the body. The same reasoning applies to call literals in the head. If such a call literal is true, the rule is automatically satisfied and one can omit it, otherwise, the call literal is removed from the head.

Definition 4. *The* call-free reduct *${}^{\sigma}P$ of a LPC $P = \langle R, \sigma : \mathcal{C}_R \rightarrow \mathit{Inst}\rangle$ are the rules $(\alpha\backslash\mathcal{C}^*_R) \leftarrow (\beta\backslash\mathcal{C}^*_R)$ where $\alpha \leftarrow \beta \in R$ and* $\bigwedge \mathsf{eval}_\sigma(\beta \cap \mathcal{C}^*_R) =$ ***true*** *and* $\bigvee \mathsf{eval}_\sigma(\alpha \cap \mathcal{C}^*_R) =$ ***false****.*[2]

For the call semantics from Table 1, assume the evaluation of σ is as in Table 2. One

[2] If a set X is empty, we assume $\bigwedge X =$ **true** and $\bigvee X =$ **false**.

Table 2. Evaluation of σ

$\mathsf{eval}_\sigma(\mathit{pizzi.res}(\mathit{8pm}))$ = **true**	$\mathsf{eval}_\sigma(\mathit{geo.near}(\mathit{pizzi}, \mathit{kine}))$ = **true**
$\mathsf{eval}_\sigma(\mathit{pizzi.res}(\mathit{10pm}))$ = **true**	$\mathsf{eval}_\sigma(\mathit{geo.near}(\mathit{ilpast}, \mathit{kine}))$ = **false**
$\mathsf{eval}_\sigma(\mathit{ilpast.res}(\mathit{8pm}))$ = **true**	$\mathsf{eval}_\sigma(\mathit{moviedb1.roman}(\mathit{kine}))$ = **true**
$\mathsf{eval}_\sigma(\mathit{ilpast.res}(\mathit{10pm}))$ = **false**	$\mathsf{eval}_\sigma(\mathit{moviedb2.roman}(\mathit{kine}))$ = **false**

can thus reserve at both 8 P.M. and 10 P.M. in *pizzi*, while only at 8 P.M. in *ilpast*. Furthermore, *pizzi* is near the movie theater, and *ilpast* is not. According to $\mathit{moviedb1}$, kine features romantic movies, contradicting $\mathit{moviedb2}$. The call-free reduct of the example contains, among others, rules

$$\mathit{plan}(\mathit{pizzi}, \mathit{8pm}, \mathit{kine}, \mathit{10pm}) \leftarrow \mathit{not}\ \mathit{otherpl}(\mathit{pizzi}, \mathit{8pm}, \mathit{kine}, \mathit{10pm})$$
$$\mathit{plan}(\mathit{pizzi}, \mathit{10pm}, \mathit{kine}, \mathit{8pm}) \leftarrow \mathit{not}\ \mathit{otherpl}(\mathit{pizzi}, \mathit{10pm}, \mathit{kine}, \mathit{8pm})$$

originating from rule p, and rules $\mathit{roman}(\mathit{kine}) \leftarrow$ and $\neg \mathit{roman}(\mathit{kine}) \leftarrow$, originating from, respectively, r_1 and r_4.

Definition 5. *An* interpretation *of a LPC* $P = \langle R, \sigma \rangle$ *is an interpretation of* ${}^\sigma P$. *An interpretation* M *of* P *is an* extended answer set *of* P *if* M *is an extended answer set of* ${}^\sigma P$.

We have 6 different extended answer sets of the example LPC:

$$\begin{aligned}
M_1 =& \{\mathit{plan}(\mathit{pizzi}, \mathit{8pm}, \mathit{kine}, \mathit{10pm}), \mathit{date}, \mathit{roman}(\mathit{kine})\}\\
M_2 =& \{\mathit{plan}(\mathit{pizzi}, \mathit{8pm}, \mathit{kine}, \mathit{10pm}), \neg \mathit{date}, \mathit{roman}(\mathit{kine})\}\\
M_3 =& \{\mathit{plan}(\mathit{pizzi}, \mathit{8pm}, \mathit{kine}, \mathit{10pm}), \neg \mathit{date}, \neg \mathit{roman}(\mathit{kine})\}\\
M_4 =& \{\mathit{plan}(\mathit{pizzi}, \mathit{10pm}, \mathit{kine}, \mathit{8pm}), \mathit{date}, \mathit{roman}(\mathit{kine})\}\\
M_5 =& \{\mathit{plan}(\mathit{pizzi}, \mathit{10pm}, \mathit{kine}, \mathit{8pm}), \neg \mathit{date}, \mathit{roman}(\mathit{kine})\}\\
M_6 =& \{\mathit{plan}(\mathit{pizzi}, \mathit{10pm}, \mathit{kine}, \mathit{8pm}), \neg \mathit{date}, \neg \mathit{roman}(\mathit{kine})\}
\end{aligned}$$

For the two possible plans – pizza at 8, movie at 10, or vice versa – the night out may be a date or not. If it is a date, one defeats $\neg \mathit{date} \leftarrow$ by the applied rule $\mathit{date} \leftarrow$. Furthermore, by constraint c, we need to have $\mathit{roman}(\mathit{kine})$ if date is in the answer set, which requires defeating $\neg \mathit{roman}(\mathit{kine}) \leftarrow$ by $\mathit{roman}(\mathit{kine}) \leftarrow$. Consequently, although two different sources ($\mathit{moviedb1}$ and $\mathit{moviedb2}$) yield contradictory information regarding the romantic nature of movies at a movie theater, a situation bound to occur frequently on the Semantic Web, the extended answer set semantics solves this by allowing for both solutions to coexist. The particular defeat mechanism makes sure this happens in a sensible way: a rule can be left unsatisfied if there is a competing applied rule.

Adding calls to programs, or, from a different perspective, wrapping different reasoners together using a logic program, amounts to reasoning that is not much worse than its worst call to a reasoner. It can be done in $\mathrm{P}^{\mathsf{comp}(\sigma)} \cup \mathrm{NP}$: either in polynomial time with an oracle of complexity the call complexity of the call semantics or in NP.

Theorem 1. *Let $P = \langle R, \sigma \rangle$ be a LPC and l a literal in R that is not a call literal. Checking whether there is an extended answer set of P containing l is in* $\mathrm{P}^{\mathsf{comp}(\sigma)} \cup \mathrm{NP}$.

Given the NEXPTIME call complexity for the night out example, checking whether there is an extended answer set containing a literal is in $\mathrm{P}^{\mathrm{NEXPTIME}} \cup \mathrm{NP} = \mathrm{P}^{\mathrm{NEXPTIME}}$, i.e. it can be done in polynomial time with an oracle in NEXPTIME (corresponding to the complexity of OWL DL instance checking).

Theorem 2. *Let $P = \langle R, \sigma \rangle$ be a LPC and l a literal in R that is not a call literal. Checking whether there is an extended answer set of P containing l is* $(\mathsf{comp}(\sigma) \cup \mathrm{NP})$-*hard.*

Approaches where input from the program can be send to the external source are not expressible in this framework, e.g. in [11] atoms calculated in the program can influence reasoning in a DL knowledge base (semantically, by adding them to the DL knowledge base). Our approach does allow for parametrized calls to sources, but the parameters must be known at compile-time before starting the computation of the answer set.

The extended answer set semantics enables resolution of conflicts. However, usually, some resolutions are more preferred than others. E.g., a particular user preference is that one rather has a quiet night out instead of a stressful date: $\neg date \leftarrow\ <\ date \leftarrow$. Moreover, not being on a date, there is no need to endure Hollywood's romantic ideals[3]:

$$\begin{array}{c} roman(kine) \leftarrow moviedb1.roman(kine) \\ roman(kine) \leftarrow moviedb2.roman(kine) \\ \hline \neg roman(kine) \leftarrow not\ moviedb1.roman(kine) \\ \neg roman(kine) \leftarrow not\ moviedb2.roman(kine) \end{array}$$

The preference between rules in the LPC, induces a natural preference relation on the rules in the call-free reduct: $\neg roman(kine) \leftarrow\ <\ roman(kine) \leftarrow$. Formally, for an order $<$ on the rules in a LPC $P = \langle R, \sigma : \mathcal{C}_R \rightarrow Inst \rangle$, we define, for rules $r_1 : (\alpha_1 \backslash \mathcal{C}_R^*) \leftarrow (\beta_1 \backslash \mathcal{C}_R^*) \in {}^\sigma P$ and $r_2 : (\alpha_2 \backslash \mathcal{C}_R^*) \leftarrow (\beta_2 \backslash \mathcal{C}_R^*) \in {}^\sigma P$,

$$r_1 {}^\sigma\!\!< r_2 \text{ iff } \alpha_1 \leftarrow \beta_1 < \alpha_2 \leftarrow \beta_2 \ .$$

Definition 6. *An* ordered program with calls (OLPC) *is a pair $P = \langle R, < \rangle$ where R is a LPC and $<$ is a strict partial order on the rules in R. An* extended answer set *of P is an extended answer set of R. An extended answer set of P is* preferred *if it is a preferred answer set of the OLP $\langle {}^\sigma R, {}^\sigma\!< \rangle$.*

Note that $\langle {}^\sigma R, {}^\sigma\!< \rangle$ is indeed an OLP, more specifically, ${}^\sigma\!<$ is a strict partial order on the rules in ${}^\sigma R$. The OLPC $\langle R, < \rangle$ defining the night out example, yields the preferred answer sets M_3 and M_6, corresponding to the preference for nights out devoid of date and romantic movie. The complexity of reasoning with OLPCs again mostly depends on the call complexity.

Theorem 3. *Let $P = \langle R, < \rangle$ be an OLPC and l a literal in R that is not a call literal. Checking whether there is a preferred answer set of P containing l is in* $\mathrm{P}^{\mathsf{comp}(\sigma)} \cup \Sigma_2^P$.

[3] The notation in modules indicates that all rules in one module, divided by a horizontal line, are more preferred than all the rules in the module above.

Theorem 4. *Let $P = \langle R, < \rangle$ be an OLPC and l a literal in R that is not a call literal. Checking whether there is a preferred answer set of P containing l is $(\mathsf{comp}(\sigma) \cup \Sigma_2^P)$-hard.*

Even though the night out example did not feature it, the heads of rules may contain calls as well. This allows a form of ontology alignment in the sense that one can enforce that ontologies should agree on some facts. E.g., $moviedb2.roman(kine) \leftarrow moviedb1.roman(kine)$ enforces that if $kine$ is a theater screening romantic movies according to $moviedb1$ then $moviedb2$ should agree. Calls in the heads of rules can, however, always be replaced by their negation in the body.

Theorem 5. *Let $\langle R, \sigma \rangle$ be a LPC with $a \leftarrow \beta \in R$ and a call literal a . Then, M is an extended answer set of $\langle R, \sigma \rangle$ iff M is an extended answer set of $\langle R', \sigma \rangle$ where $R' = (R \backslash \{a \leftarrow \beta\}) \cup \{ \leftarrow not\ a, \beta\}$.*

A similar theorem does not hold for heads that are not call literals: $a \leftarrow$ has the extended answer set $\{a\}$ while its shifted version $\leftarrow not\ a$ has no extended answer sets (one cannot motivate a since there no rules with a in the head, although the constraint demands the presence of a).

4 Preferential Reasoning on a Web of Trust

Often, the user has its particular knowledge, in the form of a program, and a sense of which calls he believes more than other calls, e.g. because (part of) one source of information is more reliable than (part of) another one. Take the LPC $\langle S, \sigma \rangle$ with S the program[4]

$$\begin{array}{rr} stock(lmby) \leftarrow & buy(S) \leftarrow ft.buy(S), nyt.buy(S) \\ stock(wtww) \leftarrow & \neg buy(S) \leftarrow not\ pdh.buy(S) \end{array}$$

with a call semantics $\sigma(ft.buy(lmby))$ = 'buy stock $lmby$ according to Financial Times' and similarly for the grounded call literals involving nyt (New York Times) and pdh (analyst Paul D'Hoore) with the stock $wtww$. Assume the evaluation of σ is as follows

$$\begin{array}{rr} \mathsf{eval}_\sigma(ft.buy(lmby)) = \textbf{false} & \mathsf{eval}_\sigma(nyt.buy(wtww)) = \textbf{true} \\ \mathsf{eval}_\sigma(ft.buy(wtww)) = \textbf{true} & \mathsf{eval}_\sigma(pdh.buy(lmby)) = \textbf{false} \\ \mathsf{eval}_\sigma(nyt.buy(lmby)) = \textbf{true} & \mathsf{eval}_\sigma(pdh.buy(wtww)) = \textbf{false} \end{array}$$

such that both the Financial Times and Paul D'Hoore discourage buying $lmby$, the Financial Times suggests buying $wtww$, while Paul would not buy $wtww$, and the New York Times suggests buying both stocks. The call-free reduct of this LPC is then

$$\begin{array}{rr} s_1 : \ stock(lmby) \leftarrow & b_f : \ \ buy(wtww) \leftarrow \\ s_2 : stock(wtww) \leftarrow & b_{p_1} : \ \neg buy(lmby) \leftarrow \\ & b_{p_2} : \neg buy(wtww) \leftarrow \end{array}$$

[4] As usual, we identify the program with its grounding.

such that we have two extended answer sets

$$N_1 = \{stock(lmby), stock(wtww), \neg buy(lmby), buy(wtww)\}$$
$$N_2 = \{stock(lmby), stock(wtww), \neg buy(lmby), \neg buy(wtww)\}$$

where b_f defeats b_{p_2}, and b_{p_2} defeats b_f respectively, corresponding to the two strategies of resolving the conflicts caused by b_f and b_{p_2}. In order to deduce the most preferred answer, we allow the user to express its belief in certain calls:

$$\{not\ pdh.buy(lmby), not\ pdh.buy(wtww)\} < \{ft.buy(lmby), ft.buy(wtww), nyt.buy(lmby), nyt.buy(wtww)\}\ ,$$

which signifies that every extended call literal in the set on the left-hand side of $<$ is more believed than any extended call literal in the set on the right-hand side, i.e. the opinion of Paul D'Hoore is valued more than the opinion of the Financial Times or the New York Times. Intuitively, this order on calls induces an order on rules. E.g. take the ground rules $b_1 : buy(wtww) \leftarrow ft.buy(wtww), nyt.buy(wtww)$ and $b_2 : \neg buy(wtww) \leftarrow not\ pdh.buy(wtww)$. We can order those rules based on the order on the call literals: we consider b_2 more preferred than b_1 since for every extended call literal in the body of b_1 that is not in the body of b_2 we have a more believed extended call literal in the body of b_2 that is not in the body of b_1. Put otherwise, for every call that b_1 needs to make in order to deduce $buy(wtww)$ and that b_2 does not make to deduce $\neg buy(wtww)$, b_2 makes a more credible call that b_1 does not make. The order on extended call literals thus induces the order $b_2 < b_1$, and a similar ordering for the grounding with $lmby$, which in turn leads to the order $\neg buy(wtww) \leftarrow\ < buy(wtww) \leftarrow$ in the call-free reduct. Consequently, the example LPOC has the preferred answer set N_2. Things get more complicated, however, if we replace, e.g., b_1 by $b_1^1 : buy(wtww) \leftarrow tmp$ and $b_1^2 : tmp \leftarrow ft.buy(wtww), nyt.buy(wtww)$. Obviously, one still prefers b_2 over b_1^1, but, now, a direct comparison based on the order on the call literals in their respective bodies does not makes sense. Instead, we look at the *trace* of both bodies, i.e. those extended call literals that must be evaluated as true in order to make the extended literals in the body true. The trace of a set of extended literals thus identifies those calls that are responsible for the truth of those literals in an extended answer set, and on which we can base the induced order on rules.

Definition 7. *Let $\langle R, \sigma \rangle$ be a LPC with call literals $\mathcal{C}_R$, $c \in \mathcal{C}_R^*$, and $l \in \mathcal{L}_R^* \backslash \mathcal{C}_R^*$. Then $c \in tr(l)$ iff for every evaluation eval_σ of σ: if M is an extended answer set of $\langle R, \sigma \rangle$ (w.r.t. eval_σ) such that $M \models l$, then $\mathsf{eval}_\sigma(c) =$ **true**.*

Furthermore, $tr(c) = \{c\}$ and $tr(\beta) = \bigcup\{tr(b) \mid b \in \beta\}$.

The trace of tmp is then $tr(tmp) = \{ft.buy(wtww), nyt.buy(wtww)\}$, i.e. in order make tmp true one needs the truth of the call literals in $tr(tmp)$. The trace of the body of b_2 is $\{not\ pdh.buy(wtww)\}$. Such that, based on those traces and the order on the call literals, we can deduce that b_2 is more preferred than b_1^1.

Definition 8. *A* program with ordered calls (LPOC) *is a pair $P = \langle R, \prec \rangle$ where R is a LPC with call literals $\mathcal{C}_R$ and $\prec$ is a strict partial order on the (extended) call literals in $\mathcal{C}_R^*$. An* extended answer set *of P is an extended answer set of R. An extended* *answer set of P is* preferred *if it is a preferred answer set of the OLPC $\langle R, < \rangle$, where, for conflicting rules $r_1 : a \leftarrow \beta_1$ and $r_2 : \neg a \leftarrow \beta_2$ in R, $r_1 \leq r_2$ iff $tr(\beta_2) \backslash tr(\beta_1) \neq \emptyset \wedge \forall c \in tr(\beta_2) \backslash tr(\beta_1) \cdot \exists c' \in tr(\beta_1) \backslash tr(\beta_2) \cdot c' \prec c$, and, for arbitrary rules $r, s \in R$, $r < s$ iff $r \leq^* s \wedge s \not\leq^* r$ where $\leq^*$ is the transitive closure of $\leq$.*

The preference order $<$ is a strict partial order such that $\langle R, < \rangle$ is indeed a LPOC.

Note that one can immediately reduce an order on knowledge sources – knowledge source Σ_1 has more authority than Σ_2 – to an order on extended call literals by grouping call literals concerning the same sources together, as we did in the stock example. An order on extended call literals instead of on sources allows for a finer granularity as it makes it possible to prefer sources for certain types of knowledge while preferring others for other types of knowledge: calls to the sports paper *L'Equipe* regarding tennis could be considered more reliable than tennis-related calls to *Le Monde*, while the opposite may be true for political subjects.

A Semantic Web agent may not always have preferences on the sources it is reasoning with, but if there is a network of agents it trusts available, it can easily learn preferences from those trusted agents. We model the Semantic Web as a pair $\langle \mathcal{K}, \mathcal{A} \rangle$ where $\mathcal{K}$ is a set of knowledge sources $\mathcal{K}$ and $\mathcal{A} = (V, E)$ is a directed graph with agents V and edges E between them. Each agent in V is defined as a LPOC, i.e. an agent has reasoning capabilities through a logic program with calls and can express preferences on its calls. Denote with $R(A)$ the sequence of agents that are reachable from A via a path in E, and assume $R(A)$ is ordered according to the trust A has in them. Thus $R(A)$ is a sequence of agents $A_1, A_2, \ldots$, such that each A_i is trusted more by A than A_{i+1} is. We thus assume that the agent resides on a web of trust, with a suitable trust metric that allows for the construction of $R(A)$ for every agent A.

For our convenience, we identify the set of sources $\mathcal{K}$ with the set of all instances of decidable decision problems d that have an associated complexity $\mathsf{comp}(d)$. E.g., the identification of a particular description logic knowledge base $\Sigma \in \mathcal{K}$ includes the set of all satisfiability checking problems w.r.t. Σ.

Take an agent $A = \langle P, \prec \rangle$ with P a simplified version of the stock example, $b_1 : buy \leftarrow ft.buy, nyt.buy$, and $b_2 : \neg buy \leftarrow not\ pdh.buy$, with call literals $ft.buy$, $nyt.buy$, and $pdh.buy$, evaluated as **true**, **true**, and **false** respectively. We assume that the agent has no preference on the two extended answer sets $\{buy\}$ and $\{\neg buy\}$ of this program, i.e. $\prec$ is empty, such that both extended answer sets are preferred. Due to the empty preference, the agent has to choose between 2 equally preferred, but contradicting, strategies. Assuming the agent is part of network of agents it trusts, it can try to find out what the trusted agents think of its call literals. E.g., assume that agent A is connected to agents $A_1 = \langle P_1, \prec_1 \rangle$, $A_2 = \langle P_2, \prec_2 \rangle$, and $A_3 = \langle P_3, \prec_3 \rangle$ with preferences defined as follows:

$$\begin{array}{rr} not\ pdh.buy \prec_1 ft.buy & not\ pdh.buy \prec_3 nyt.buy \\ lat.buy \prec_1 ft.buy & ft.buy \prec_3 lat.buy \\ ft.buy \prec_2 not\ pdh.buy & \end{array}$$

Thus, agent A_1 prefers Paul D'Hoore's advice as well as the Los Angeles Times's advice over that of the Financial Times, agent A_2 holds an opposite view and prefers the Financial Times over Paul D'Hoore, and agent A_3 prefers Paul's advice over the New

York Times's and has more believe in the Financial Times than in the Los Angeles Times. We do not specify the programs of those agents since we are only interested to learn preferences for agent A from the preferences its trusted agents have – for A it does not matter how the trusted agents deploy those preferences.

In order to let agent A construct its preferences based on this web of agents, we assume its reachable agents are ranked according to trustworthiness: $R(A) = A_1, A_2, A_3$, such that A_1 is the agent that A trusts the most and A_3 the agent that it trusts the least. Considering the preference of A_1, A only retains $not\ pdh.buy \prec_1 ft.buy$: combining this preference with A's own preference, a strict partial order on the call literals of P can be constructed. The other preference of A_1 involves $lat.buy$ which is of no concern to agent A since it is not a call literal in P.

Moving to agent A_2, second in the line of trust, A ignores $\prec_2$: it contradicts the order already constructed in A with the more trusted agent A_1. Finally, A ignores the preference in $\prec_3$ involving $lat.buy$, but it updates its preference with $not\ pdh.buy \prec_3 nyt.buy$. This results in an updated agent $A' = \langle P, \prec' \rangle$ with $not\ pdh.buy \prec' ft.buy$, and $not\ pdh.buy \prec' nyt.buy$. This order on call literals induces then the order $b_2 < b_1$ such that $\{\neg buy\}$ is the preferred answer set of the updated agent A'.

Definition 9. *For an agent $A = \langle P, \prec \rangle$ in $\mathcal{A}$, let $R(A) = \langle P_1, \prec_1 \rangle, \langle P_2, \prec_2 \rangle, \ldots$ The* updated agent *of A is $A' = \langle P, \prec' \rangle$ where $\prec' = (\prec \cup \bigcup_{i=1} B_i)^*$ with*

1. $B_i \subseteq \prec_i$,
2. $\forall c_1 \prec_i c_2 \in B_i \cdot c_1, c_2 \in \mathcal{C}_P^*$,
3. $(\prec \cup \bigcup_{j=1}^{i} B_j)^*$ *is a strict partial order,*
4. B_i *is a maximal set satisfying* $1.$, $2.$, *and* $3.$

Intuitively, the agent updates its own preference $\prec$ with maximal subsets of preferences of trusted agents, and this according to the order of trust. Condition $2.$ ensures that only preferences on call literals of the agent's own program P are considered, and condition $3.$ ensures that only those preferences of $\prec_i$ are retained that, when added to the accumulated preference and transitively closing the result, one still has a strict partial order. The latter only amounts to checking irreflexivity since transitivity is entailed by taking the transitive closure. Condition $4.$ forces $\prec'$ to consider as much preferences as possible from each preference $\prec_i$.

The updated $\prec'$ is a strict partial order on call literals such that the updated agent A' is a LPOC, and we can compute preferred answer sets of an agent by computing the preferred answer set of its updated version that takes into account the web of trust.

Definition 10. *Let A be an agent in $\mathcal{A}$. The preferred answer set of A is the preferred answer set of the updated A'.*

In order to be able to compute the updated agent for an agent A, we assume that $R(A)$ is finite. Since the Semantic Web with software agents is finite this sounds like a reasonable restriction. However, due to the sheer amount of envisaged agents on the Semantic Web, it is unlikely that feasible reasoning with all connected agents is possible. A possible strategy in overcoming this problem is to add a bound on the number of trusted agents in the sequence $R(A)$.

In considering an agent as a logic program, we neglected a lot of the machinery involved in agent definitions. E.g., in the IMPACT System [1] an agent consists of two parts: software code and a semantic wrapper consisting of a message manager, an action module, and a meta-knowledge module. In [12], the theory and implementation of the action module is described, with, among others, code call atoms that are able to call software, and agent programs that express the choices for actions. E.g., $\mathbf{O}(send\_note(Person)) \leftarrow \mathbf{Do}(run\_audit(Person))$, indicates that if one is executing the audit run, one is obliged to send a note. Conflict resolution in [12] amounts to allowing defeat of the meta-rule "if $\mathbf{O}\alpha$ then $\mathbf{Do}\alpha$", which says that if action α is obliged then one should execute it. This type of behavior can be simulated under our extended answer set semantics by introducing the ordered rules $\mathbf{Do}(\alpha) \leftarrow \mathbf{O}(\alpha) < \neg\mathbf{Do}(\alpha) \leftarrow \mathbf{O}(\alpha)$, thus minimizing defeat of the meta-rule. Moreover, our preference relation between rules allows for more fine-grained types of conflict resolution as showed in this section.

5 Conclusions and Directions for Further Research

We devised and discussed a logic programming based framework for agents on the Semantic Web, where agents are capable of expressing preferences on the rules or on the call literals in their knowledge. Those preferences enabled the resolution of conflicts with the most preferred solution. In case an agent has no preferences but is part of a web of trusted agents, we showed how the agent can replenish its own preferences based on the preferences of trusted agents.

The preferred answer set semantics from Section 2, i.e. without calls, was implemented by the OLPS solver [26], available at http://tinf2.vub.ac.be/olp/. For a given OLPC, i.e. a program with calls and an order on those rules, different plug-ins are envisaged to be written, depending on the type of desired calls. Such a plug-in's main task would be to execute the decision problem associated with a particular call, e.g. check the satisfiability of a concept with the FACT [21] DL reasoner, and subsequently calculate the call-free reduct and the reduced order on this reduct, which are then to be fed to OLPS.

References

1. K. Arisha, T. Eiter, S. Kraus, F. Ozcan, R. Ross, and V. S. Subrahmanian. IMPACT: Interactive Maryland Platform for Agents Collaborating Together. *IEEE Intelligent Systems*, 14(2):64–72, 1999.
2. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003.
3. C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.
4. S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference, 2004.
5. K. Van Belleghem, M. Denecker, and D. De Schreye. A Strong Correspondence between DLs and Open Logic Programming. In *Proc. of ICLP'97*, pages 346–360, 1997.
6. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

7. L. Ding, L. Zhou, and T. Finin. Trust Based Knowledge Outsourcing for Semantic Web Agents. In *Proc. of the 2003 IEEE/WIC International Conference on Web Intelligence*, 2003.
8. F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. AL-log: Integrating Datalog and Description Logics. *J. of Intell. and Cooperative Information Systems*, 10:227–252, 1998.
9. T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. Planning under Incomplete Knowledge. In *Proc. of CL 2000*, volume 1861 of *LNCS*, pages 807–821. Springer, 2000.
10. T. Eiter, G. Gottlob, and H. Veith. Modular Logic Programming and Generalized Quantifiers. In *Proc. of LPNMR*, pages 290–309, 1997.
11. T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining Answer Set Programming with DLs for the Semantic Web. In *Proc. of KR 2004*, pages 141–151, 2004.
12. T. Eiter, V. S. Subrahmanian, and G. Pick. Heterogeneous Active Agents, I: Semantics. *Artif. Intell.*, 108(1-2):179–255, 1999.
13. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Proc. of ICLP'88*, pages 1070–1080, Cambridge, Massachusetts, 1988. MIT Press.
14. Y. Gil and V. Ratnakar. Trusting Information Sources One Citizen at a Time. In *Proc. of International Semantic Web Conference (ISWC 2002)*, pages 162–176, 2002.
15. J. Golbeck and J. Hendler. Inferring Reputation on the Semantic Web. In *Proc. of WWW 2004*. ACM, 2004.
16. J. Golbeck, B. Parsia, and J. Hendler. Trust Networks on the Semantic Web. In *Proc. of Cooperative Intelligent Agents 2003*, 2003.
17. B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proc. of WWW 2003*, pages 48–57, 2003.
18. James Hendler. Agents and the Semantic Web. *IEEE Intelligent Systems Journal*, 16(2), 2001.
19. S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Nonmonotonic Ontological and Rule-based Reasoning with Extended Conceptual Logic Programs. In *Proc. of ESWC 2005*, number 3532 in LNCS, pages 392–407. Springer, 2005.
20. S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Preferential Reasoning on a Web of Trust. Technical report, Vrije Universiteit Brussel, Dept. of Computer Science, 2005. http://tinf2.vub.ac.be/˜sheymans/tech/aspc-tech.ps.gz.
21. I. Horrocks. The FaCT system. In *Proc. of Tableaux'98*, number 1397, pages 307–312. Springer-Verlag, 1998.
22. I. Horrocks and P. Patel-Schneider. Reducing OWL Entailment to Description Logic Satisfiability. *J. of Web Semantics*, 1(4):345–357, 2004.
23. U. Hustadt, B. Motik, and U. Sattler. Reducing $\mathcal{SHIQ}^-$ Description Logic to Disjunctive Datalog Programs. FZI-Report 1-8-11/03, Forschungszentrum Informatik (FZI), 2003.
24. V. Lifschitz. Answer Set Programming and Plan Generation. *Journal of Artificial Intelligence*, 138(1-2):39–54, 2002.
25. B. Motik, R. Volz, and A. Maedche. Optimizing Query Answering in Description Logics using disjunctive deductive databases. In *Proc. of KRDB'03*, pages 39–50, 2003.
26. D. Van Nieuwenborgh, S. Heymans, and D. Vermeir. An Ordered Logic Program Solver. In *Proc. of PADL 2005*, number 3350 in LNCS, pages 128–142. Springer, 2005.
27. C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.
28. M. Richardson, R. Agrawal, and P. Domingos. Trust Management for the Semantic Web. In *Proc. of ISWC 2003*, pages 351–368. Springer-Verlag, 2003.
29. R. Rosati. Towards Expressive KR Systems Integrating Datalog and Description Logics: Preliminary Report. In *Proc. of DL'99*, pages 160–164, 1999.
30. D. Van Nieuwenborgh and D. Vermeir. Preferred Answer Sets for Ordered Logic Programs. In *Proc. of JELIA 2002*, volume 2424 of *LNAI*, pages 432–443. Springer, 2002.
31. Matthias Wagner, Thorsten Liebig, Olaf Noppens, Steffen Balzer, and Wolfgang Kellerer. Towards Semantic-based Service Discovery on Tiny Mobile Devices.