Multiple metareasoning agents for flexible query-answering systems

Jonas Barklund, Stefania Costantini*,
Pierangelo Dell’Acqua and Gaetano A. Lanzarone*

Uppsala University
Computing Science Department
Box 311, S-751 05 Uppsala, Sweden
Phone: +46–18–18 25 00
Fax: +46–18–51 19 25

Abstract

We show through a number of examples how it is possible to express beliefs of agents and communication among agents by employing a first-order logic language with metalogical facilities. We argue that (i) such a language is suitable to characterise rational agents in different application areas, and (ii) the interplay between metalogical reasoning and different modalities of communications allows us to obtain systems with a high degree of flexibility.

*Dipartimento di Scienze dell’Informazione, Università degli Studi di Milano
1 Introduction

The perception of a user about “flexibility” of interaction with a computer system can be in general based on several aspects, like the followings. **Syntactic flexibility**: the system is able to answer a question even in case the syntax of the question does not coincide with the internal representation; also, the system is able to adapt the syntax to the needs/skills of the particular user. **Reasoning flexibility**: the system is able to employ non-classical or non-deductive forms of reasoning, like for instance analogy, induction, etc. The user may possibly be asked to validate the conclusions. **Knowledge-level flexibility**: according to a classification of the users and/or of the questions, the system is able to use different portions of knowledge and/or different deduction strategies. An important aspect is the ability of the system to adapt the modalities of interaction to the particular user.

In this paper we claim, and demonstrate by means of a number of examples, that a formalism equipped with metalogic and reflection capabilities allows to easily model (by means of the meta-level) and implement (by means of reflection) the desired degree of flexibility. We also claim that a main feature for building a really flexible system is the possibility of modeling agents, which are able to interact and to communicate (by means of common knowledge and/or explicit message-passing). Agents can incorporate different behaviour and/or different knowledge, so as to implement, with their global behaviour, a complex and really flexible knowledge-based system. The computer based agents in multiagent systems should have a metareasoning capability in order to exhibit a rational behaviour when interacting with each other and with human agents.

With ‘having a metareasoning capability’ we mean that they are capable of reasoning about their own beliefs and inferences, as well as those of other agents. This metareasoning capability should also reach further than one level, as in “He believes that I believe that he is lying”, although it needs not extend to arbitrarily many levels.

With ‘interacting’ we primarily mean agents asking questions and agents telling propositions, but it could also involve other acts, such as exchanging money for services.

With ‘rational behaviour’ in this context we mean that an agent should be able to behave differently, according to its own model of the other agents involved in the interaction. I.e., it should exhibit some or all of the following characteristics and others in the same spirit:

- An agent $A$ should not communicate to an agent $B$ information that $A$ believes $B$ already believes.

- An agent $A$ should not make crucial decisions based on information received from an agent $B$, if agent $B$ is not trusted.
An agent $A$ should be able to pinpoint an incorrect inference drawn by an agent $B$, if $A$ has sufficient information about the beliefs of $B$.

Our argumentation will consist of a collection of examples of how expressing the beliefs of an agent using a first-order logic language with a metalogical extension facilitates achieving a rational and flexible behaviour when interacting with other agents. It will be clear that the metalogical extensions simplify achieving the desired behaviour, even though in principle they could be achieved using only object-level language. (This is similar to the observation how programming in a high-level programming language makes it possible to develop applications that in principle could have been written in assembly language although that would not have been feasible in practice.)

The relevance for query-answering systems is immediate: it is useful to view a query-answering system as a collection of agents that interact with other external (typically human) agents. By incorporating metalevel reasoning, we obtain query-answering systems that behave more rationally and flexibly towards users.

2 Logic languages with metalogical extensions

The examples in this paper are written in two similar extensions of Horn clause logic. The example of Section 5 is expressed in Alloy [3, 4], a language designed for specifying systems of interrelated theories. The examples of Sections 3 and 4 are written in Reflective Prolog [6, 7].

Alloy

Syntactically we can characterize the Alloy language as follows:

- A program consists of a set of statements on the form $\tau \vdash \pi$ or $\tau_1 \equiv \tau_2$, where each $\tau$, $\tau_1$ or $\tau_2$ is a theory term and each $\pi$ is a program clause. A statement $\tau \vdash \pi$ expresses that $\pi$ belongs to the theory $\tau$ while a statement $\tau_1 \equiv \tau_2$ expresses that the theories $\tau_1$ and $\tau_2$ have the same theorems.

- The atoms are extended with expressions on the form $\neg \tau \vdash \pi$ and $\tau_1 \equiv \tau_2$, where $\tau$ etc. and $\pi$ are as above. These atoms make it possible to express in the language itself statements about the theorems of theories.

- Theory terms are terms that denote theories.

The terms are extended with expressions on the form $\tau_1 \circ \tau_2$, which are theory terms if $\tau_1$ and $\tau_2$ are. A theory denoted by $\tau_1 \circ \tau_2$ is the “view” in $\tau_1$ of the theory $\tau_2$. 

2
The terms are also extended with expressions on the form $^\gamma E$ for each term or formula (atom, goal or program clause) $E$. The latter terms denote the “quoted” expressions and we call them *name terms*.

- Finally, in any expression $^\gamma \cdots ^\gamma$, a variable might be written with a dot above it. It means that the variable is not part of the represented formula but rather part of the representation, which is then nonground. It is only meaningful to substitute a name term for such a variable.

An inference system (although not the most efficient conceivable) for Alloy is given elsewhere [4].

An Alloy program specifies a theory or a collection of theories that typically will be (or can be thought of as) the beliefs of one of more agents. The nonlogical axioms of a theory $A$ need not be given by an explicit listing but may instead be given implicitly as the solutions for $p$ of a statement $B \vdash ^\gamma C \vdash p^\gamma$, provided that the theorems of $A$ coincide with (or contain) those of $B \bowtie C$, i.e., the statement $A \equiv B \bowtie C$ holds.

The example in Section 5 specifies a single theory *Traffic* (corresponding to $B$ above) that specifies the theorems of various other theories, each corresponding to the beliefs of the driver of some car, or the driver’s beliefs about other drivers. The theory *Traffic* could thus be the program of some agent supervising a crossing between roads, reasoning about drivers that approach it.

**Reflective Prolog**

Reflective Prolog is a metalogic programming and knowledge representation language. It has three basic features. First, it assumes as given the specification of a full *naming mechanism* which allows the representation of metaknowledge in the form of *metalevel clauses*. The naming mechanism must obey certain requirements, but in principle may vary depending on the needs of a specific application domain. We can say that the language takes the naming mechanism as a “parameter”. In the following, the name of an expression $A$ is conventionally indicated with 'A'. (For sake of simplicity we use a different notation from [1, 2].) If, for instance, we want to express properties (i.e., metaknowledge) of an expression of the object-language (i.e., knowledge) such as $p(a, X)$, we have to employ a name of that expression, represented here as $p(a, 'X)$, where 'X stands for the name of what will be the value of $X$.

The second feature concerns the possibility of specifying *metaevaluation clauses* that are clauses defining the predicate symbol *solve*. Metaevaluation clauses allow to declaratively extend the meaning of other predicates. Suppose, for example, that we want to express the fact that an object $a$ satisfies all the relations in a given class *class*. If we want to formalise that statement by object-level clauses, then for every predicate $p$ in *class* we have
to write the clause $p(a)$. Instead, by using a metaevaluation clause we can formalise our statement as

$$solve(P(a)) \leftarrow belongs_to(P, class)$$

where $P$ is a metavariable ranging over the names of predicate symbols. Last, a form of logic reflection which makes this extensions effective both semantically and procedurally [6]. Semantically, this has been achieved by introducing the notion of reflective models. They are models in usual sense but satisfy an additional requirement that an atom $A$ belongs to the model if and only if the atom $solve(A)$ belongs to the model. Procedural semantics of Reflective Prolog programs is based on an extension of SLD-resolution that allows shifting between the object level and the metalevel of the language. Informally, the extended SLD-resolution is able to use clauses with conclusion $solve(A)$ to resolve a goal $A$, and vice versa clauses with conclusion $A$ to resolve $solve(A)$.

By means of metalevel features, a good deal of syntactic flexibility in query-answering can be easily reached, as shown by the following example based on taxonomies.

$$human(man)$$
$$animal(human)$$
$$man(john)$$

Assume that a user may want to ask queries in a different syntax, like for instance:

$$?- is\_a(man, human)$$

This is allowed by the following metaevaluation clause

$$solve(is\_a(Obj, Class)) \leftarrow solve(Class(Obj))$$

A different kind of flexibility results from implementing, again by means of metaevaluation clauses, non-classical or non-deductive forms of reasoning, like for instance analogical reasoning, various forms of induction, etc. In the context of knowledge bases however, modularization capabilities are essential for providing a really flexible view of the knowledge base to the different kinds of users interacting with the system. In sections 3 and 4, we show how the language can be enriched with the possibility of defining multiple communicating agents, and then demonstrate through a number of examples how the interplay between metalogical reasoning and different modalities of communications allows us to obtain systems with a high degree of flexibility.
3 Communication-based reasoning

The ability to represent agents and multiagent co-operation is central to many AI applications. In the context of communication-based reasoning, the interaction among agents is based on communication acts. In particular, every agent can ask other agents questions in order to solve a given problem.

Within the logic programming paradigm, an approach to communication-based reasoning has been proposed by Costantini et al. [5]. The main idea of that approach is to represent agents and communication acts by means of theories and reflection principles, respectively. Thus, theories formalize knowledge of agents, while reflection principles characterize possible kinds of interaction among agents.

Every agent has associated with it a theory represented by a finite set of clauses prefixed with the corresponding theory symbol. In the following, for specifying that a clause \( A \leftarrow B_1, \ldots, B_n \) belongs to a theory \( T \), we use the notation \( T : A \leftarrow T : B_1, \ldots, T : B_n \), or equivalently \( T : A \leftarrow B_1, \ldots, B_n \). A program is then defined as a set of theories. Hereafter, we use the notation \( 'A \) to indicate the name of the syntactic expression \( A \), whatever naming convention is adopted.

Communication acts are formalized by means of inter-theory reflection axioms based on the (binary) predicate symbols \( \text{tell} \) and \( \text{told} \).

\[
T : \text{told}(S,'A) \leftarrow S : \text{tell}(T,'A)
\]

Its intuitive meaning is that every time an atom of the form \( S : \text{tell}(T,'A) \) can be derived from a theory \( S \) (which means that agent \( S \) wants to communicate proposition \( A \) to agent \( T \)), the atom \( T : \text{told}(S,'A) \) is consequently derived in the theory \( T \) (which means that proposition \( A \) becomes available to agent \( T \)).

The objective of this formalization is that every agent can specify, by means of clauses defining the predicate \( \text{tell} \), the modalities of interaction with the other agents. These modalities can vary with respect to different agents or different conditions as the following examples show. (More elaborate examples of the use of agents in Reflective Prolog can be found in [5].)

- An agent \( A \) tells an agent \( B \) a thing it can prove and lies to an agent \( C \).

\[
A : \text{tell}(B,'p(a)) \leftarrow p(a)
A : \text{tell}(C,\neg p(a)) \leftarrow p(a)
\]

- An agent \( A \) tells an agent \( B \) a thing it can prove within some resource limitations,

\[
A : \text{tell}(B,'X) \leftarrow \text{limited\_prove}(X)
\]

where the predicate \( \text{limited\_prove} \) incorporates the desired limitations.
• An agent $A$ trusts an agent $B$ but distrusts an agent $C$.

$A : p(X) \leftarrow \text{reliable}(B), \text{told}(B, p(X))$

$A : p(X) \leftarrow \text{told}(C, \neg p(X))$

4 Query-answering in knowledge bases

In the framework of multiple metareasoning agents, agents can be used to formalize not only knowledge bases and the corresponding user interfaces, but also external users. In fact, the approach to inter-agent communication introduced above allows us to formalize a variety of interactions among agents depending on the application context. This is shown in the next examples.

• An agent $A$ tells the others whatever it can prove about a predicate $p$.

$A : \text{tell}(T, p(X)) \leftarrow p(X)$

• An agent $A$ tells a group of agents whatever it can prove about a predicate $p$, and another group of agents whatever it can prove about predicate $q$.

$A : \text{tell}(T, p(X)) \leftarrow \text{group1}(T), p(X)$

$A : \text{tell}(T, q(X)) \leftarrow \text{group2}(T), q(X)$

• An agent $A$ tells the others whatever it can prove about a predicate $p$ in a certain module $M$.

$A : \text{tell}(T, p(X)) \leftarrow \text{demo}(M, p(X))$

• An agent $A$ tells the agents in a certain group whatever it can prove about a predicate $p$ in a module $M$ related to that group.

$A : \text{tell}(T, p(X)) \leftarrow \text{group1}(T), \text{module}(\text{group1}, M), \text{demo}(M, p(X))$

• Different agents may have different reasoning power and different knowledge bases.

$A_1 : \text{tell}(T, p(X)) \leftarrow \text{demo}_{A_1}(M_{A_1}, p(X))$

$A_2 : \text{tell}(T, p(X)) \leftarrow \text{demo}_{A_2}(M_{A_2}, p(X))$

• An agent $A$ may ask other agents to answer a query according to which group the request is coming from.

$A : \text{tell}(T, Q) \leftarrow \text{group1}(T), \text{server}(\text{group1}, S_1), \text{told}(S_1, Q)$

$A : \text{tell}(T, Q) \leftarrow \text{group2}(T), \text{server}(\text{group2}, S_2), \text{told}(S_2, Q)$
The agent $A$ may also pass to the server the information about the original agent asking the query. This is achieved by means of an optional third argument to $tell/told$. In this way the server can decide locally about what kind of answer to give according to where the original request comes from.

$$
A : tell(T,Q) \leftarrow group1(T), server(group1, S1), told(S1,Q,T) \\
A : tell(T,Q) \leftarrow group2(T), server(group2, S2), told(S2,Q,T)
$$

Users can be modeled as agents interacting with the knowledge base. Agents representing users can be divided into categories, or groups, corresponding for instance to: (i) their access privileges; (ii) the portion of the knowledge base they need to access; (iii) the view of the knowledge base they need to have. Then, also the format of the answer can be adapted to the needs or the skill of the particular user.

An agent implementing a user interface, can route questions other agents, associated to a specific portion of the knowledge base, and/or to different kind of answers to the same questions, according to the group the user belongs to.

The specification of the agents is in the usual style of Horn clause logic programming, since all these modalities of interactions are defined as metalogic axioms of the above form. They are applied (like Horn clauses) to $tell/told$ subgoals, and, by means of reflection, result in the context-switching and information exchange among the involved agents.

Below we sketch the specification of an intelligent tutoring system able to provide students with exercises to solve, concerning specific topics they are interested in.

The agent $interface$ defines the user interface of this system, which receives requests for new exercises from the users. According to a table that associates students to topics, $interface$ asks the agent managing that topic for the exercise, also specifying who is the user issuing the request.

The agent $maths$ is an example of an agent managing a topic, i.e., mathematics, that selects an exercise from a suitable library according to the level of the user (beginner/expert).

$$
interface : tell(\text{User,'Exercise}) \leftarrow interested\_in(\text{User,'Topic}), \\
told(\text{Topic,'Exercise, User}) \\
interface : interested\_in(\text{anne,'maths}) \\
interface : interested\_in(\text{george,'maths}) \\
interface : interested\_in(\text{dan,'computer\_science})
$$
As an example of programming multiple agents that reason about each other, consider the traffic problem illustrated in Fig. 1. In this example, communication is not important; rather it is how the agents solve the problem without communicating (although using much common knowledge).

Three cars are simultaneously approaching a four-way crossing. There are no other signs or traffic lights, so the rule (which is common knowledge) is that drivers should give way to cars coming on their right side. Using a simple application of this rule, we obtain that car A can pass, while cars B and C must wait, because they give way to some car on their right side. However, the driver of car C could instead reason that car B must wait, because the driver of car B will see car A on her right entering the crossing and give way to it. Hence, the driver of car C might conclude that he can safely pass.

This form of reasoning is probably illegal in most countries, but there is empirical evidence that drivers often reason this way and it is interesting to note that it is naturally modelled as a case of multilevel metareasoning.
We program this example in Alloy as a theory Traffic, which will contain our theorems about the situation and about the knowledge of drivers.

The following statement encodes the problem of the driver of car C.

\[
\text{Traffic} \vdash \left( D(C, \text{South} \right) \vdash \text{Pass}([D(A, \text{North}), D(B, \text{East}), D(C, \text{South})]) \right) \]

The theory Traffic is where our reasoning about the drivers will take place.

Let a term \( D(x, y) \) denote the driver of car \( x \) coming from direction \( y \).

Each theory \( \text{Traffic} \circ D(x, y) \) represents (our view of) the beliefs of the driver \( D(x, y) \). Each theory \( \text{Traffic} \circ D(x_1, y_1) \circ D(x_2, y_2) \) represents (our view of) the beliefs that driver \( D(x_1, y_1) \) has about driver \( D(x_2, y_2) \), and so on.

A theorem \( \text{Pass}(z) \) in a theory \( \cdots \circ D(x, y) \) would mean that the driver in question believes that she can pass a crossing in which she sees the cars listed in \( z \). Similarly, a theorem \( \text{Wait}(x) \) would mean that she believes that she has to stop.

The first two clauses are interesting, because they help us to encode a form of common knowledge (calling it group belief seems more appropriate but we will use the standard term):

\[
\text{Traffic} \vdash \text{Driver}(D(x, y), D(x, y)) \\
\text{Traffic} \vdash \text{Driver}(D(x, y) \circ p, d) \leftarrow \text{Driver}(p, d)
\]

Every atom on the form \( \text{Driver}(D(x_1, y_1) \circ \cdots \circ D(x_n, y_n), D(x, y)) \) is a theorem in Traffic. For example, we can derive \( \text{Traffic} \vdash \text{Driver}(D(C, \text{South}), D(C, \text{South})) \) and \( \text{Traffic} \vdash \text{Driver}(D(C, \text{South}) \circ D(B, \text{East}) \circ D(A, \text{North}), D(A, \text{North})) \). Note that each such theorem is about a theory term encoding some driver’s view of some driver’s view of… some driver’s beliefs, and the ultimate driver in such a chain. We can use the predicate Driver in Traffic for expressing that something should be believed by every driver and that every driver should believe that other drivers believe so, etc., arbitrarily deep.

The following three clauses define the common knowledge of drivers:

\[
\text{Traffic} \vdash \left[ \text{Pass}(c) \leftarrow \text{Not-in-crossing}(x_1, c) \right] \leftarrow \right. \\
\text{Driver}(t, d) \land t_1 \land x_1 \land \text{Names } x \land \text{Gives-way-to}(d, x) \\
\text{Traffic} \vdash \left[ \text{Wait}(c) \leftarrow \text{In-crossing}(x_1, c) \land \right. \\
\left. x_2 \land \text{Names } x_1 \land c_1 \land \text{Names } c \land \right. \\
\left. \text{Pass}(c_1) \right] \leftarrow \right. \\
\text{Driver}(t, d) \land t_1 \land x_1 \land \text{Names } x \land \text{Gives-way-to}(d, x) \\
\text{Traffic} \vdash \left[ \text{Pass}(c) \leftarrow \text{In-crossing}(x_1, c) \land \right. \\
\left. x_2 \land \text{Names } x_1 \land c_1 \land \text{Names } c \land \right. \\
\left. \text{Wait}(c_1) \right] \leftarrow \right. \\
\text{Driver}(t, d) \land t_1 \land x_1 \land \text{Names } x \land \text{Gives-way-to}(d, x)
\]

9
The first clause says that any driver will reason: if there is no car approaching from a direction to which I must give way, then I may pass.

The second clause says that any driver will reason: if there is a car approaching from a direction to which I must give way, and I believe that the driver of that car will reason that he can pass, then I must wait.

The third clause says that any driver will reason: if there is a car approaching from a direction to which I must give way, but I believe that the driver of that car will reason that he must wait, then I can pass anyway.

The next four clauses of Traffic simply determine who must yield to whom.

\[ \text{Traffic} \vdash \text{Gives-way-to}(D(\text{North}), D(\text{West})) \]
\[ \text{Traffic} \vdash \text{Gives-way-to}(D(\text{West}), D(\text{South})) \]
\[ \text{Traffic} \vdash \text{Gives-way-to}(D(\text{South}), D(\text{East})) \]
\[ \text{Traffic} \vdash \text{Gives-way-to}(D(\text{East}), D(\text{North})) \]

The predicates In-crossing and Not-in-crossing are essentially list membership and nonmembership predicates; these predicates are part of the common knowledge of drivers.

\[ \text{Traffic} \vdash \neg \text{In-crossing}(x, [\_]) \leftarrow \text{Driver}(t, d) \land t \text{ Names } t \]
\[ \text{Traffic} \vdash \neg \text{In-crossing}(x, [c]) \leftarrow \text{In-crossing}(x, c) \leftarrow \text{Driver}(t, d) \land t \text{ Names } t \]
\[ \text{Traffic} \vdash \neg \text{Not-in-crossing}(x, [\_]) \leftarrow \text{Driver}(t, d) \land t \text{ Names } t \]
\[ \text{Traffic} \vdash \neg \text{Not-in-crossing}(x, [y|c]) \leftarrow x \neq y \land \text{Not-in-crossing}(x, c) \leftarrow \text{Driver}(t, d) \land t \text{ Names } t \]

A full proof of the original statement (1) is rather long, but involves proving the following key statements:

\[ \text{Traffic} \circ D(\text{C, South}) \circ D(\text{B, East}) \circ D(\text{A, North}) \vdash \text{Pass}([D(\text{A, North}), D(\text{B, East}), D(\text{C, South})]) \]
\[ \text{Traffic} \circ D(\text{C, South}) \circ D(\text{B, East}) \vdash \text{Wait}([D(\text{A, North}), D(\text{B, East}), D(\text{C, South})]) \]
\[ \text{Traffic} \circ D(\text{C, South}) \vdash \text{Pass}([D(\text{A, North}), D(\text{B, East}), D(\text{C, South})]) \]
\[ \text{Traffic} \vdash \neg D(\text{C, South}) \vdash \text{Pass}([D(\text{A, North}), D(\text{B, East}), D(\text{C, South})]) \]

6 Conclusions

We have shown several examples of how to specify at the metalevel reasoning patterns of agents that interact with other agents. Most of them have been very straightforward to express and it should be easy to see that the metalevel is the appropriate level for specifying them. Then, a first claim is that the use of metalogic languages makes it easy to reach a good degree of flexibility.
Also, we should have convinced the reader that the metalevel gives a system the appropriate perspective for specifying its own model of the external users, and its own model of the interactions with the users. The model on which the interactions are based can rely on a classification of both the users, and the kind of questions they are allowed to ask. The answer can be “tuned” accordingly, both syntactically and in contents.

References


