Processing temporal and atemporal declarative knowledge in metalogic programming

Gaetano A. Lanzarone and Alessandro Provetti

Abstract

This paper concerns the application of logic programming to temporal reasoning, seen as a relevant instance of McCarthy's approach to contextual reasoning, where contexts are treated as first-class objects. We introduce reflection as a suitable device for this kind of reasoning; in particular, it makes possible to decontextualize atoms $holds(f, t)$ and use f as an atomic formula in combination with other non-temporal formulae of the outer context. We show these features by applying Reflective Prolog, a metalogic language based on logical reflection, to a simple form of Sergot and Kowalski's Event Calculus. This redefinition results in more expressive calculi with dynamic contextualization/ decontextualization, a well-defined semantics and an effective procedural counterpart. Therefore, we take reflection as a promising theoretical principle for representing and computing contexts, and show that its embodiment in Reflective Prolog -an implemented system- is useful for temporal reasoning applications.

Keywords: Knowledge Representation, Meta and Higher-Order Logic Programming.

1 Motivations

Several AI projects are concerned with defining ontologies and building knowledge bases that are large, general-purpose, sharable and reusable, i.e., presumably, some knowledge bases of this sort describe time-dependent information together with atemporal properties of entities of the domain. The need may arise to combine temporal reasoning on time-varying information and logical (i.e., atemporal) reasoning with pieces of knowledge that, *for the purpose at hand,* can be viewed as always holding.

Relations have properties that depend on user's conceptualization of the world. For instance, the relation *above* represents the transitive closure of relation *on* because of language conventions and of space conceptualization. Also, time relations such as *before* and *after* are antisymmetric and irreflexive. Actions taking place in certain locations may involve invariant properties of relations and locations (e.g., in a house, which is in a city, which is in a state...). It is desirable to represent both kinds of knowledge and combine both kinds of reasoning within a uniform language, proof procedures and model-theoretical semantics. In addition, it may be useful to state axioms relating temporal to atemporal properties:

Both authors are with D.S.I. - Università di Milano. Via Comelico 39, Milan. I-20135 Italy. {*lanzaron,provetti*}*@dsi.unimi.it*

- if a relation always holds, then it does so over any time interval/instant: $Kb \models \phi \Rightarrow Kb \models holds\_at(< \phi >, i)$;
- also, if a relation holds on an interval, then an equivalent relation should hold thereof.

An example of *hybrid* reasoning is a set of queries posed regarding an industrial project: A person is assigned to a project only if i) she is an expert of the topic (and this expertise does not change in the lifetime of the project) *and* ii) she is free from other projects at the moment. *This is the type of queries we are trying to address in this paper.*

The paper is organized as follows. Section 2 is a survey on the issue of reification in contextual and temporal reasoning. In Sec. 3 we recall the Event calculus. In Sec. 4 and 5, we first show how this language can accommodate the definition of EC (and contextual reasoning in general) by exploiting its metalanguage features; we then discuss other aspects of temporal reasoning that are not expressible in EC at the same level of generality achieved in RP; finally, we give examples of reasoning with mixed contextual and non-contextual propositions.

2 Contexts and reified temporal formalisms

McCarthy [McC93] discusses the problem of introducing and formalizing contexts as first-class objects. The basic formula is

$$ist(p, c)$$

where p is a proposition and c is a context. The goal is to *lift* axioms for static situations to contexts in which situations change, and the motivation is that [*ibid*]:

> [...] this is necessary if the axioms are to be included in general common sense databases that can be used by any program needing to know about the phenomenon covered but which may be concerned with other matters as well.

The problem is that of introducing a formalization which, on the one hand, is sufficiently expressive for knowledge representation and has useful mathematical properties, and, on the other hand, is suitable for efficient processing of knowledge. In this paper we consider the metalogical language Reflective Prolog, a logic programming Horn-clause language extended with reification and logical reflection. It has a complete declarative semantics and is a fully implemented system. We argue that a language with such capabilities is a suitable tool for formalizing contextual reasoning.

Reification is a well-known representation tool which allows higher-order sentences to be represented in first-order logic as metasentences. This means that contexts and contextual sentences can be declaratively defined at the metalevel. Logical reflection is a mathematical tool which can be given a procedural import so as to allow both object-level and metalevel knowledge to be exploited in proofs.

Reification and logical reflection are the key features for overcoming the difficulties of representing both contextual and non-contextual sentences in the same framework. The case study of this paper is temporal reasoning, a relevant form of contextual reasoning, and we will focus on the temporal formalism Event Calculus, proposed by Sergot and Kowalski for reasoning about time, action and change within logic programming.

Temporal reasoning can by now be considered a well-established subfield of AI (see [ShoMcD87] for an introduction and overview). Nevertheless, the debate is still open on both the ontological and the logical aspects of representing temporal information. Here we are only interested in the technical aspects of representing and deriving temporal information in logical form. In this respect, we will take our stance in the debate about the reified vs. non-reified approach [Gal91, BTK91]. We will argue that this dispute disappears if a proper kind of reification is adopted, and even more so if reflection is taken into account, as a powerful means for dynamic reification/unreification of propositions.

In order to do so, let us first review the issues about reification as they appear in the literature. As pointed out by several authors, temporal information can be represented in a logical formalism in several ways. The general problem is how to associate a temporal and atemporal expressions in assertions.

The simplest way is the 'method of temporal arguments' of [Hau87] where one parameter for time is added to an ordinary logic predicate expressing a proposition type. As Shoham comments in [Sho87], there is nothing technically wrong with this representation, but it accords no special status to time. Thus, it has been dismissed in favor of the so-called reified approach.

In the reified approach, the symbols of both the temporal and the atemporal component appear as arguments to some encompassing symbol, as in $True(p, i)$ or $Holds(p, i)$, where p is a proposition type and i is a time point or time interval. Reified propositions were featured by the first and most influential approaches to temporal reasoning, i.e. Allen's Interval Calculus [All84] and McDermott's Temporal Logic [McD82]. The problem is what logical status to assign to these symbols: reified propositions formally are considered as terms (being arguments), but (semantically) are thought of as representing relations. As Shoham noticed [Sho87], both [All84] and [McD82] suffer of semantical problems, i.e. neither give their sentences a clear meaning.

In [Sho87] Shoham proposed another approach, aimed at giving a semantics to the temporal formalism, i.e., expressions: $TRUE(p, i)$, where $TRUE$ is neither a relation nor a modal operator, but a context symbol affecting the semantics: the interpretation of a relation p is determined not only by the symbol itself but also by the interval i[1].

The non-reified proposal of Bacchus et al. [BTK91] maintains that Shoham is right in seeking a well-defined semantics, but being his model-theoretic semantics non standard, it has no available proof theory, and thus there is no reasoning procedure. Therefore, a non-reified logic is advocated, resulting in a two-sorted first-order logic, one sort for temporal and one sort for atemporal arguments of predicates, without resorting to reification. The benefits of this logic are that the classical semantics and proof theory are available [*ibid*].

Another quest for unreification is from Galton in [Gal91]. He proposes a general method for converting type-reification into token-reification, so as to regard terms occurring in temporal context as individuals, without stepping outside the bounds of first-order logic. In particular, he applies this method to Kowalski and Sergot's Event Calculus.

However, both [BTK91] and [Gal91] admit that some expressive power is lost in the unreified approach. As an instance, the assertion *'effects precede their causes'* can be expressed in reified but not in unreified logics, since it requires quantification over propositions. This is particularly important also for expressing *frame axioms*, which are necessary for temporal projection.

To summarize, reified approaches suffer from insufficient semantic power, while non-reified or unreified approaches suffer from insufficient expressive power.

[1] See [Mon92] for a discussion of this issue within EC proper.

In order to illustrate our proposal, aimed at overcoming both limitations, we consider here a simple version of the Event Calculus (EC), introduced by Sergot and Kowalski [KowSer86] and later discussed -among others- by [ChiMon93, Kow92, KowSad97, Ser90].

Its definition shows some of the characteristics usually present in the reified approach. Atomic formulae of the form $Holds(f, i)$ are used to express that a fluent f Jholds over an interval of time i. In the logic programming framework, where meta-programming is a common technique, it comes natural to recognize the higher-order nature of these propositions, and to try to represent them at the meta-level. In fact, [Kow92] observes on Situation Calculus:

> Thus we write
>
> $$Holds(possess(Bob\ Book1)\ S0)$$
>
> instead of the weaker but also adequate
>
> $$Possess(Bob\ Book1\ S0).$$
>
> In the first formulation, $possess(Bob\ Book1)$ is a term which names a relationship. In the second, $Possess(Bob\ Book1\ S0)$ is an atomic formula. Both representations are expressed within the formalism of first-order classical logic. However, the first allows variables to range over relationships whereas the second does not. If we identify relationships with atomic variable-free sentences, then we can regard a term such as $possess(Bob\ Book1)$ as the name of a sentence. In this case $Holds$ is a metalevel predicate, [...] .

This recognition, however, had not led to a full treatment of names as first-class objects in the representation of EC, which instead relied on the ambiguity between symbols and their names. In the following, we will introduce a representation of EC in Reflective Prolog (RP), that emphasizes the metalinguistic aspects which Kowalski points to. We will try to highlight:

- the ability and the elegance of RP in capturing the meta-level aspects of a reified formalism, viz. EC;
- the desirable features resulting from using the full metalevel architecture of RP and, in particular, how it is possible to separate the treatment of time (dates) from the representation of the domain-relevant objects;
- how it is possible to quantify over fluent names and have amalgamated propositions with EC-components (the fluents) intermixed with components of the outer program (atoms of traditional logic programming).

3 The Event Calculus

Event Calculus (EC) assumes an ontology of events, considered more primitive than time, and of fluents, i.e., *partial* descriptions of the world that are true during certain intervals of time and change their truth value in response to the occurrence of events over time. Maximal and convex intervals of validity are called periods. Given a period[2] for the fluent

[2]In the following the notation is Prolog-like, i.e. predicates, constant functions and constant symbols begin with lower case letters while variable symbols are upper case. For the sake of simplicity, dates will always be natural numbers and compared by means of the infix predicate $<$ with its standard meaning.

f, no super-interval of the period exists where f holds continuously. A period is bounded by events that make the fluent considered true in the world -at the start- and false in the world -at the end. Events, then, are associated with the fluents that they affect by means of relations describing initiation (making true in the world) and termination (making false in the world). Syntactically, events are represented by means of constants ($e1, e2, \ldots$) and fluents by first-order terms (though they may be intended as relations). In this respect, EC is first-order.

By an *Event Calculus system* it is usually intended a logic program/deductive database made up of:

i) a database of records of events;

ii) a set of context-dependent predicate definitions that link types of events to fluents;

iii) a set of rules for deriving fluents and their periods. A form of temporal ordering on events must be specified, either using absolute dates, i.e. time-stamping events, or using a relative ordering on events, e.g. instantiating facts like $before(e1, e2)$.

An EC system can deduce periods of validity of fluents and which fluents hold at a certain time; in its simpler version, it works under the assumption that events are recorded into the database in the same order as they happened in the world. The general case (events recorded in any order) is discussed in Section 3.2.

3.1 The Axioms of Event Calculus

In this section we introduce the Event Calculus axioms as a logic program. We will show a small example of querying a database of events in order to present the general axioms that are used in the rest of the paper[3].

Suppose there is a departmental database which contains records of promotions, retirements etc. In our terminology the changes in the department staff modeled by the database are called events. Events are unambiguously identified by describing the type of change they bring about, their date etc. The usual representation of events in EC, borrowed from case semantics, is a set of instances of binary predicates, where each event is labeled by a constant. This is a simple and easily modifiable form that allows for partial descriptions of events. Updates can be seen as additive only, albeit the case of deleting wrong information is still present. For example:

$happens\_at(e1, 1985).$
$actor(e1, mario).$
$act(e1, promotion).$
$relation(e1, professor).$

These events in the departmental database initiate and terminate periods of time for which a person has a certain rank. The fact that Mario is a professor, written $rank(mario, professor)$, is an example of a fluent: an assertions descriptive of the world whose *truth-value* is defined over time. The link between events and fluents is established by the definition of initiates and terminates:

[3]For a fuller discussion of EC and its motivations refer to the original proposal in [KowSer86] and later developments, e.g. in [Ser90, Kow92, KowSad97]; relevant extension of the formalism are found in [Mon92, CCM95]; some database applications are in [DiaPat97, Sri95].

$$initiates(E, rank(Name, NewRank)) \ :- \ actor(E, Name), \\ act(E, promotion), \\ relation(E, NewRank).$$

$$terminates(E, rank(Name, OldRank)) \ :- \ actor(E, Name), \\ act(E, demotion), \\ relation(E, OldRank).$$

The user's specification of the domain remains defined as the set of fluents mentioned in the initiates/terminates relations. Let us proceed to define predicate *holds,* needed for querying the database about periods of validity of a certain fluent, e.g. for what period one has been professor:

$$? - holds(rank(mario, professor), P).$$

Holds is the topmost predicate of Event Calculus and is defined as follows (using [,] as a grouping function):

$$(EC1)\ holds(F, [Ts, Te]) \ :- \ happens\_at(E1, Ts), \\ initiates(E1, F), \\ happens\_at(E2, Ts), \\ terminates(E2, F), \\ Ts < Te, \\ not\ terminated(Ts, F, Te).$$

The predicate *terminated* is used to make sure that during the period (Ts, Te) there are no other events that affect F[4].

$$(EC2)\ terminated(Ts, F, T) \ :- \ happens\_at(Ebet, Tbet), \\ Ts \leq Tbet, \\ Tbet < T, \\ terminates(Ebet, F).$$

An alternative way for querying the database is to check the holding of a fluent at a specified date t, with a query like: $? - holds\_at(rank(mario, professor), Ts, t)$. The second parameter returns the date of the event that made the fluent true as an additional information.

$$(EC3)\ holds\_at(F, Ts, T) \ :- \ happens\_at(E, Ts), \\ initiates(E, F), \\ Ts < T, \\ not\ terminated(Ts, F, T).$$

3.2 Dealing with Mutually Exclusive Fluents

A further aspect of EC is the treatment of mutually exclusive fluents. For them, Kowalski and Sergot propose to declare such exclusivity, and use it to stop incorrect derivations based on default persistence. This is necessary when events are recorded in an order different from their happening[5]. Suppose for instance that the two positions of research assistant and professor are incompatible, and suppose also that it is known only that:

[4]Procedurally, in order to minimize the interpretations of *terminates*, *happens_at* is used to generate a candidate event to be tested against the temporal bounds; the formulation of $EC2$ and $EC3$ takes this aspect into account. This and other issues on the efficiency of query-answering are discussed in [ChiMon93].

[5]See [Sri95] for a discussion on EC for dealing with such updates in temporal deductive databases.

($e1$) Mario was appointed Professor in October 1990
($e2$) Mario was hired as a Research Assistant, effective from August 1991
($e3$) Mario resigned from Professor in November 1991

Then, EC should not derive (by abuse of notation) $holds(rank(mario, professor), [aug91, nov91])$, for otherwise it would result that Mario is holding a professor and a *'RA'* position at the same time. To cope with such situation, it is sufficient to replace the predicate *terminated* in rules EC1-EC3 with *broken,* defined as follows:

$(EC4)$ $broken(Ts, F, T) :- happens\_at(Ebet, Tbet),$
$Ts < Tbet,$
$Tbet < T,$
$initiates(Ebet, Fbet),$
$exclusive(F, Fbet).$

$(EC5)$ $broken(Ts, F, T) :- happens\_at(Ebet, Tbet),$
$Ts < Tbet,$
$Tbet < T,$
$terminates(Ebet, Fbet),$
$exclusive(F, Fbet).$

The first rule for *broken* looks for an event that initiates an incompatible fluent. In such case the interpretation of *broken* succeeds and therefore the holds call fails. The second rule checks whether there is evidence of an incompatible fluent started later than the one under consideration, thus breaking default persistence. Along with *initiates* and *terminates, exclusive* is dependent on the domain. Hence, defining *exclusive* is another way to express domain knowledge. In our example:

$exclusive(professor, res\_assistant).$
$(*)$ $exclusive(res\_assistant, professor).$

In the example, the interpretation of rule EC4 against $(*)$ blocks the derivation of an incorrect period for $rank(mario, res\_assistant)$. For dealing with the case of contrasting information on the same fluent we include the rule:

$(EC6)$ $exclusive(F, F).$

4 The rôle of reification

In this Section we reformulate the example of Section 2 in Reflective Prolog; the changes in the syntax are rather limited, even if they already yield advantages w.r.t. the original version. In Section 5 a substantial departure from standard EC is taken, in order to achieve the significant improvements in expressivity. To begin with, this is an example of an event description in the new syntax:

$happens\_at(e1, 1985).$
$actor(e1, mario).$
$act(e1, promotion).$
$relation(e1, < professor >).$

The above sentences may coexist in the same theory with non-contextual sentences, i.e. those not depending on time (as quoted in [McC93], Quine called them *eternal sentences*), for instance:

$likes(mario, english\_literature).$
$italian(mario).$
$rank(< professor >).$

Notice that the only change so far is in using names (angular brackets) for relation symbols occurring as arguments of other relations or fluents. The definition of *initiates*, *terminates*, *holds*, and *holds_at* are now given as metalevel sentences[6]:

$$\begin{aligned} initiates(E, \#F(\text{"}Q\text{"})) \quad :- \; & rank(\#F), \\ & actor(E, Q), \\ & act(E, promotion), \\ & relation(E, \#F). \end{aligned}$$

$$\begin{aligned} terminates(E, \#F(\text{"}Q\text{"})) \; :- \; & rank(\#F), \\ & actor(E, Q), \\ & act(E, retire), \\ & relation(E, \#F). \end{aligned}$$

¿From the above rules it is possible for instance to derive $initiates(e1, < professor > (\text{"}mario\text{"}))$ where $< professor > (\text{"}mario\text{"})$ is the reification (i.e., a term which acts as the name) of the atomic formula $professor(mario)$. The predicates *holds* and *holds_at* are now re-defined at the metalevel, exploiting the metalevel negation mechanism. The positive rules, given below, assume the persistence of every fluent for which an initiating event is recorded.

$$\begin{aligned} holds(\#F(\$Args), Ts, Te) \quad :- \; & happens\_at(E1, Ts), \\ & initiates(E1, \#F(\$Args)), \\ & happens\_at(E2, Te), \\ & Ts < Te, \\ & terminates(E2, \#F(\$Args)). \end{aligned}$$

$$\begin{aligned} holds\_at(\#F(\$Args), Ts, T) \; :- \; & happens\_at(E, Ts), \\ & Ts < T, \\ & initiates(E, \#F(\$Args)). \end{aligned}$$

Let us stress the possibility to quantify over meta-variables ranging over names of fluents (see in the example the quantification over fluents of the kind *rank*); we can therefore define rules for initiation/termination at a level of generality not found in the literature on EC. The straightforward outcome is the possibility of querying about (the names of) relations an individual is involved into: $? - solve(< holds > (\#F(\text{"}mario\text{"}, \$Args), \text{"}Ts\text{"}, \text{"}Te\text{"}))$, with answer, in the example, $\#F =< professor >$. Again, this kind of querying was not possible in standard EC. The negative part of the definition (consisting of metarules defining *solve_not*) specify the particular cases in which persistence does not hold. As explained in Section 3.1, this is the case whenever a fluent is terminated or broken.

$$solve\_not(< holds > (\text{"}\#F(\$Args)\text{"}, \text{"}Ts\text{"}, \text{"}Te\text{"})) \quad :- \; broken(\#F(\$Args), Ts, Te).$$

$$solve\_not(< holds\_at > (\text{"}\#F(\$Args)\text{"}, \text{"}Ts\text{"}, \text{"}T\text{"})) \; :- \; broken(\#F(\$Args), Ts, T).$$

[6]To better understand the new axioms, please refer to the Appendix and the original definitions in Section 3.1.

$$broken(\#F(\$Args), Ts, Te) :- happens\_at(E2, T*), Ts \leq T*, T* < Te, initiates(E2, \#Q(\$A)), exclusive(\#F(\$Args), \#Q(\$A)).$$

$$broken(\#F(\$Args), Ts, Te) :- happens\_at(E2, T*), Ts < T*, T* < Te, terminates(E2, \#Q(\$A)), exclusive(\#F(\$Args), \#Q(\$A)).$$

Negative conditions are automatically verified on every atomic formula which is derivable from the positive part of the program; they play the role of integrity constraints since their success forces the given formula to fail. Notice that extra negative conditions on *holds* and *holds_at* can be specified by adding new negative metarules, without modifications of those already existing. The definition of *exclusive* is now given at the meta-level, coherently with the kind of knowledge (predication of predicates) that this relation formalizes :

general statements:	*domain dependent definitions:*
$exclusive(\$Q, \$Q).$	$exclusive(< professor > (\$X), < res\_assistant > (\$X)).$
$symmetric(< exclusive >).$	$rank(< professor >).$
	$rank(< res\_assistant >)$

The symmetry of exclusive may be declaratively and computationally treated by the following auxiliary rule:

$$solve(\#P(\$X, \$Y)) :- symmetric(\#P), solve(\#P(\$Y, \$X)).$$

This rule is automatically applied via reflection, and has the effect of exchanging the arguments of the given atomic formula and retrying to prove it. For instance, $exclusive(< res\_assistant > (\$X), < professor > (\$X))$ which is not derivable from the given database becomes $exclusive(< professor > (\$X), < res\_assistant > (\$X))$, which instead succeeds)[7]. In a similar way to *holds* and *holds_at*, we may define predicates $holds\_since(Fluent, Time)$, and $holds\_now(Fluent)$, by exploiting the *holds_at* definition[8]:

$$holds\_since(\$P, T) :- date(Tnow), holds\_at(\$P, T, Tnow).$$

$$solve\_not(< holds\_since > (\text{"}\$P\text{"}, \text{"}T\text{"})) :- broken(\$P, Ts, T).$$

$$holds\_now(\$P) :- date(Tnow), holds\_at(\$P, Ts, Tnow).$$

5 The rôle of reflection

In the previous Section we have stressed that RP is sufficiently flexible for defining eternal sentences without forcing them into a temporal form[9]. We can now show that RP

[7]Here we have the flavor of how the metalevel can treat not only EC, but also other kinds of properties of relations. These may also take place in combination, if necessary. The actual number of levels used in the computation can be deduced only by the ground instantiation of the program

[8]Here *Date(T)* is a predefined predicate which instantiates T to the current time unit.

[9]In [Sri95], for instance, eternal sentences are treated and represented in the form: $holds(eternal\_sentence, *, *)$ where $'*'$ stays for infinite.

reflection capabilities also allow *relatively eternal sentences* and *relative decontextualization* in the sense of [McC93]; thus introducing the possibility of taking a sentence out of its temporal context. For instance, assume that we want to find all the professors, regardless of the time when they were promoted; of course, we may use a query: $?-holds\_now(<Professor>("X"))$. Relative decontextualization implies instead the possibility of querying with $?-professor(X)$. This can be achieved by means of the straightforward RP rule:

$solve(\#P(\$A)) : -holds\_now(\#P(\$A)).$

Using this rule, $professor(mario)$ is derived as follows:

$?-professor(mario) \Rightarrow$ fails $\Rightarrow$ logical reflection $\Rightarrow$ automatic upward reflection
$?-solve(<professor>("mario")) \Rightarrow$ resolution step
$?-holds\_now(<professor>("mario")) \Rightarrow$ succeeds after some steps $\Rightarrow$ yes.

A rule which uses $professor(X)$ as a relatively eternal sentence is for instance:

$salary(X, 3000) : -professor(X).$

Vice versa, *default contextualization* can also be defined. In temporal reasoning it may be assumed that non-contextual sentences hold over any interval of time:

$solve(<holds>("\$P","T1","T2")) : -solve(\$P).$

This rule allows for instance to derive $holds(<likes>("mario","english\_literature"), T1, T2)$ for every $T1$ and $T2$.

5.1 Time as a context (sketch)

Another relevant application of the metalevel and reflection capabilities of RP is substitutivity of fluents within contexts. The temporal context is a particular example of a transparent context [McC93], and substitutivity may be easily defined as a property of the context as in the example below:

$context(<holds>, temporal).$ $context(<holds\_at>, temporal).$
$context(<holds\_since>, temporal).$ $context(<holds\_now>, temporal).$
$transparent\_context(temporal).$

$equivalent(<professor>, <instructor>).$
$symmetric(<substitutable>).$
$substitutable(\#R, \#S) : -equivalent(\#R, \#S).$

$$solve(\#P("\#Q(\$A)", \$Other\_args)) :- context(\#P, C), transparent\_context(C), substitutable(\#Q, \#G), solve(\#P("\#G(\$A)", \$Other\_args)).$$

This solve rule is yet again an auxiliary inference rule which is automatically applied via reflection whenever needed. It is defined on predicates $\#P$ which have an atomic formula $\#Q(\$Args)$ among their arguments. It applies whenever $\#P$ is a relation concerning a transparent context. The effect is that $\#P$ of $\#Q(\$A)$ is derivable whenever $\#P$ of $\#G(\$A)$ is, and the two predicates $\#Q$ and $\#G$ are substitutable, e.g., $holds\_now(<instructor>("mario"))$ can be derived from our example program.

6 Final remarks

Logic programming is a more or less obvious vehicle for representing temporally-scoped knowledge, witness to it the literature from logic programming conferences and journals. We have shown that Meta-logic programming is a useful technique for these tasks, offering a simple but powerful representation style, and efficient query-answering mechanisms. Strictly speaking, the Reflective Prolog approach allows to express

i) queries quantified over fluent relations, e.g. $? - holds(\#P(a))$;

ii) time-independent properties of fluents (e.g., symmetry), and

iii) *hybrid* statements involving temporal and atemporal conditions.

The next step in our research will be the evaluation of our approach against realistic examples, e.g. [DiaPat97] where several contextual aspects are present, thus showing how the metalogic approach allows: i)several contexts to coexist in the same theory, and ii) these contexts to be explicitly defined as transparent or opaque. What is important, in conclusion, is that reification plus logical reflection provide a tool for designing and experimenting various forms of contextual reasoning, within the same application domain.

References

[All84] J. Allen, 1984. *Towards a General Theory of Action and Time.* Artificial Intelligence 23:123–154.

[BTK91] F. Bacchus, J. Tenenberg and J. Koomen, 1991. *A Non-reified Temporal Logic.* Artificial Intelligence 52:87–108.

[CCM95] I. Cervesato, L. Chittaro and A. Montanari, 1995. *A Modal Calculus of Partially Ordered Events in a Logic Programming Framework.* Proc. of ICLP'95, pp. 299–313.

[ChiMon93] L. Chittaro and A. Montanari, 1993. *Reasoning about discrete processes in a logic programming framework.* Proc. of GULP'93, pp. 407–421, Mediterranean Press.

[CosLan94a] S. Costantini and G.A. Lanzarone, 1994. *A Metalogic Programming Approach: Language, Semantics and Applications.* Journal of Experimental and Theoretical Artificial Intellligence, 6:239–287.

[CosLan94] S. Costantini and G.A. Lanzarone, 1994. *Metalevel Negation in Non-monotonic Reasoning.* Methods of Logic in Computer Science:1.

[DiaPat97] O. Díaz and N. Paton, 1997. *Stimuli and business policies as modeling constructs: their definition and validation through the event calculus.* In Proc. of CAiSE'97, pp. 33–46.

[Gal91] A. Galton, 1991. *Reified Temporal Theories and How to Unreify Them.* Procs. of 12th IJCAI, pp. 1177–1182.

[Hau87] B. Haugh, 1987. *Non-standard Semantics for the Method of Temporal Arguments.* In Proc. of 10th IJCAI, pp. 449–455.

[Kow92] R. Kowalski, 1992. *Database Updates in the Event Calculus.* Journal of Logic Programming 12(1-2):121–146.

[KowSad97] R. Kowalski and F. Sadri, 1997. *Reconciling the Event Calculus with the Situation Calculus.* Journal of Logic Programming 31(1-3):39–58.

[KowSer86] R. Kowalski and M. Sergot, 1986. *A Logic-Based Calculus of Events.* New Generation Computing 4, Springer Verlag, pp 67–95.

[McC93] J. McCarthy, 1993. *Notes on Formalizing Context.* Proc. of AAAI Symposium on Logical Formalizations of Commonsense Reasoning, pp 133–146. See also proc. of IJCAI '93.

[McD82] D. McDermott, 1982. *A Temporal Logic for Reasoning about Processes and Plans.* Cognitive Science 6:101–155.

[Mon92] A. Montanari, E. Maim, E. Ciapessoni and E. Ratto, 1992. *Dealing with Time Granularity in the Event Calculus.* Proc. of FGCS'92, pp. 702-712.

[Ser90] M. Sergot, 1990. *(Some topics in) Logic Programming in AI.* Lecture Notes of the GULP Advanced School on Logic Programming. Unpublished.

[Sho87] Y. Shoham, 1987. *Temporal Logics in AI: Semantical and Ontological Considerations.* Artificial Intelligence 33:89–104.

[ShoMcD87] Y. Shoham and D. McDermott, 1987. *Reasoning, Temporal.* In Shapiro S.C. (ed.), Encyclopedia of Artificial Intelligence, pp. 870–875.

[Sri95] S. Sripada, 1995. *Efficient implementation of the event calculus for temporal database application.* Proc. of ICLP'95, pp. 99–113.

A Reflective Prolog

The metalogic programming and knowledge representation language Reflective Prolog [CosLan94a, CosLan94] is an extension of the Horn-clause language which allows the declarative representation of knowledge and metaknowledge at distinct but connected levels, uniformly in the same language.

Knowledge and metaknowledge are integrated by the same inference mechanism, which provides an automatic interaction between levels (i.e., it is not necessary to specify control information about when to change the level of the inference process or what kind of knowledge to exploit in a proof). Reflective Prolog implements three innovative features:

- a full naming mechanism for terms and atoms that allows the representation of metaknowledge (in the following, the name of a term/atom A is indicated by A');
- the possibility of specifying *metaevaluation clauses* (which are the clauses defining the distinguished predicates *solve* and *solve_not*) that allow to declaratively extend/restrict the meaning of the other predicates;

- a form of *logical reflection* which makes these extensions/restrictions effective [CosLan94a] yet keeping the semantics very similar to that of standard Horn-clause language.

In fact, in order to exploit both object-level knowledge and metaknowledge in proofs, RP does not use new inference rules, and does not rely on an explicit representation of provability. Instead, the RP has a clear declarative semantics that leaves the underlying logic unchanged, but on the one hand implicitly extends the program with the *reflection axiom*

$$A :- solve(A')$$

and on the other hand restricts the set of accepted interpretations to those which allow level communication: if A belongs to the model then also $solve(A')$ does (see [CosLan94a] for a constructive characterization of models).

Procedural semantics for positive RP programs is based on RSLD-Resolution, an extension of the well-known SLD-Resolution. Informally, a goal A can be resolved by a clause with conclusion A (as usual), but also by a clause with conclusion $solve(A')$ *(upward reflection)*; vice versa a goal $solve(A')$ can be resolved by a clause with conclusion $solve(A')$ (as usual), but also by a clause with conclusion A *(downward reflection)*, which indeed is searched as first. The built-in predicate $ref(A, "A")$ is used in programs to extract the name of terms or conversely to pass from a name to the object it names, whenever this is possible.

A novel form of negation, metalevel negation, is defined by extending RSLD-Resolution to NRSLD-Resolution fro programs using the distinguished predicate *solve_not*: if A succeeds, $solve\_not(A')$ is attempted; if it succeeds, A is forced to failure, otherwise success of A is maintained. Among the minimal models the selected one is the one that never entails A if it entails $solve\_not(A')$ [CosLan94].

In order to present some examples, it is useful to summarize RP naming mechanism and procedural features. The name of a constant or variable c is $"c"$; the name of a predicate symbol p is $< p >$, of a function symbol f is $\{f\}$; the names of a term $f(a1, ..., an)$ and atom $p(a1, ..., an)$ are the name terms $f(a1', ..., an')$ and $< p > (a1', ..., an')$ respectively. The name of a clause $A :- Body$ is $A' :- Body'$. There are four kinds of variables: object variables (syntax V) to denote object-level terms; predicate metavariables (syntax $\#V$) to denote names of predicate symbols; function metavariables (syntax $\%V$) to denote names of function symbols; general metavariables (syntax $\$V$) to denote any metalevel term. Negation as failure is allowed in RP programs, since it can be explicitly implemented by means of metalevel negation [CosLan94]. For the sake of readability, let us adopt the following convention; for rules of the kind:

$$P(\ldots, "V", \ldots) :- \ldots, Q(\ldots, V, \ldots), \ldots.$$

the call $ref("V", V)$, needed for casting the variable name $"V"$ appearing in the head into the object variable V actually used in the body, is omitted.