# Formal Consistency Verification of Deliberative Agents with respect to Communication Protocols

Jaime Ramírez and Angélica de Antonio

Technical University of Madrid
Madrid, Spain
{jramirez, angelica}@fi.upm.es
http://decoroso.ls.fi.upm.es

**Abstract.** The aim of this paper is to show a method that is able to detect inconsistencies in the reasoning carried out by a deliberative agent. The agent is supposed to be provided with a hybrid Knowledge Base expressed in a language called CCR-2, based on production rules and hierarchies of frames, which permits the representation of non-monotonic reasoning, uncertain reasoning and arithmetic constraints in the rules. The method can give a specification of the scenarios in which the agent would deduce an inconsistency. We define a scenario to be a description of the initial agent's state (in the agent life cycle), a deductive tree of rule firings, and a partially ordered set of messages and/or stimuli that the agent must receive from other agents and/or the environment. Moreover, the method will make sure that the scenarios will be valid w.r.t. the communication protocols in which the agent is involved.

## 1 Introduction

The purpose of this paper is to show a method to verify the consistency of the reasoning that a deliberative agent can perform. We assume the agent to comprise a knowledge base (KB) expressed in a knowledge representation formalism called CCR-2.

The CCR-2 formalism is valid to represent hybrid KBs that combine production rules with hierarchies of frames. This formalism allows us to represent non-monotonic reasoning, uncertain reasoning, and arithmetic constraints in the rules.

We assume that the agent whose reasoning is checked needs to carry out a reasoning process for deciding its next action according to its goals. The agent's knowledge can fall into three different categories: *acquired knowledge*, *innate knowledge* or *deduced knowledge*. The acquired knowledge is made up of acquired facts, that is, information coming from its perception or requested to other agents; the innate knowledge is made up of knowledge that the agent knows since the beginning of its life; and the deduced knowledge is formed by the facts deduced by firing rules. It is clear that, as the reasoning process evolves,

the agent may obtain contradictory acquired facts from different sources w.r.t. previously acquired facts. In this case, the new knowledge would replace the obsolete knowledge. However, the agent should not be allowed to deduce a set of contradictory facts from the acquired facts and the innate facts.

The proposed method finds scenarios in which the agent would deduce an inconsistency. A scenario consists of a description of the initial agent's state (in the agent life cycle), a deductive tree of rule firings, and a partially ordered set of messages and/or stimuli (expressed as schemas) that the agent must receive from other agents and/or the environment to achieve the execution of the deductive tree. A scenario permits the execution of a deductive tree of rule firings that will deduce a set of semantically contradictory facts. We assume the agent's state to be a set of innate facts, acquired facts (from the sources mentioned above) and/or deduced facts, that is, it is a Fact Base (FB). Basically, the partially ordered set of messages and/or stimuli schemas, included as part of a scenario, will represent precedence dependencies between the messages/stimuli required in the reasoning. This set will be checked w.r.t. the communication protocols in which the verified agent is involved, so as to warrant the precedence dependencies can be satisfied by the specification of the communication protocols.

Some methods or tools designed to detect inconsistencies in a Knowledge Base System (KBS) (mostly rule-based systems) build a model of the KBS (Graph, Petri Net, etc.), and execute the model for each valid input, in order to identify possible inconsistencies during the reasoning process. This approach in many cases turns to be computationally very costly. Thus, we decided to adopt another approach in which the starting point is one of the inconsistencies that might be possibly deduced by the verified KBS, and the goal is to compute a description of the scenarios in which the KBS included in the agent would deduce that inconsistency. This approach takes some ideas from the ATMS designed by de Kleer (1), since it uses the concept of label as a way to represent a description of a set of FBs. Other methods for verifying rule-based systems that follow a similar approach were proposed in (2) (3) (4) (5) (6) (7) (8).

Section 2 explains some points related to the agent's KB and inconsistencies that are verified by this method, and the hypotheses that will be assumed in the operation of the method. In section 3 it is described how this method specifies the way in which an agent deduces an inconsistency, if possible. In section 4, the procedure for detecting an inconsistency is explained, and in section 5, a small example of application is shown. We end with some conclusions about our work, and some future works that will be derived from this work.

## 2 Scope

Our method receives as inputs a CCR-2 KB (the agent's KB), a classification of the possible facts that the agent can manage, an Integrity Constraint (IC) to be checked, and a set of communication protocol specifications.

CCR-2 (also called GKR) (9) supports the representation of production rules and a high number of object types in the FB: frame classes and instances, re-

lationships, propositions, attribute values and attribute identifiers. A rule's antecedent in CCR-2 is a Disjunctive Normal Form (DNF) formula made up of literals. A *literal* is an atom, a negated atom or a linear arithmetic inequation over attribute values and/or certainty factors. An *atom* states something about some object in the FB. In CCR-2 a rule's consequent contains a list of actions that can modify the state of an object, create or destroy objects while executing the KB system included in the agent. This last characteristic allows us to represent some types of non monotonic reasoning. As it is possible to declare variables as relationships and propositions in the rules, the antecedent of a rule is a second order logic formula. Nevertheless, the actions of the rules can not change the type of a relationship or a proposition, therefore CCR-2 supports a limited representation of the second order logic. Moreover, uncertain reasoning can be represented in CCR-2 by associating certainty factors to attribute values, to tuples in a relationship or to propositions.

The CCR-2 KBs can use two kinds of management of the negation: closed world assumption (CWA) or 3-valued logic. The kind of negation management determines: when a fact can be considered true or false; what is the effect of the actions; how the facts and actions can be chained during the KBS execution; and which pairs of actions are contradictory. For instance, in the 3-valued logic there are three truth values: true, false and unknown; while a fact will be false if its negation appears in the FB, a fact will be unknown if neither it nor its negation appear in the FB; moreover, the action $Add(\neg p)$ deduces the fact $\neg p$, and the pair of actions $Add(p)$ and $Add(\neg p)$ are contradictory. It must be highlighted that the action $Add(\neg p)$ cannot be employed under CWA.

The rules are assumed to execute with forward chaining or backward chaining under conflict set resolution. The rules are structured in groups whose activation or inhibition is controlled by metarules. When a rule is fired, we assume the sequential execution of all the actions belonging to the consequent of the rule.

We assume that two kinds of facts can appear during the agent's execution: *static facts* and *dynamic facts*. A static fact is a fact whose truth value changes neither from true to false nor from false to true during the reasoning process, whereas the truth value of a dynamic fact actually may change those ways. In this sense, acquired facts and deduced facts will be dynamic facts. Moreover, facts representing innate knowledge are assumed to be static. The method needs to know both whether a literal is static or dynamic, and whether a literal is acquired, innate or deduced, so a classification must be provided.

### 2.1 Defining Inconsistencies: Integrity Constraints

An IC defines a consistency criterion over input data, output data or input and output data. The IC form is:

$$\exists x_1 \in T_1 \exists x_2 \in T_2 ... \exists x_n \in T_n \exists() x_{n+1} \in T_{n+1} \exists() x_{n+2} \in T_{n+2} ... \exists x_{n+m} \in T_{n+m}$$
$$A \Rightarrow \bot$$

where $A$ is a second order logic formula in DNF that includes conditions over whatever types of CCR-2 objects. Each literal in $A$ has an associated scope,

which specifies whether the literal is related to input data (acquired literal or innate literal), or output data (deducible literal). For the variables in A, two kinds of quantifiers can be employed: the existential quantifier (with the classical meaning) and the restricted existential quantifier (denoted as $\exists()x$).

> An IC $\exists x \in T(A(x) \Rightarrow \bot)$ is violated if at least one object in the class $T$ that is included in the FB satisfies the conditions imposed over the variable $x$ in the formula $A$.
> An IC $\exists()x \in T(A(x) \Rightarrow \bot)$ is violated if every object in the class $T$ that is included in the FB satisfies the conditions imposed over the variable $x$ in the formula $A$ and only those conditions.

This semantics for the restricted existential quantifier permits the detection of knowledge gaps. Lets see an example of an IC with a restricted existential quantifier:

$$\exists()x \in PATIENT$$
$$Is\_Ill(x, FLU), (x.Fever = high) \Rightarrow \bot$$

Clearly, having a high fever is not enough to deduce that a patient has flu. So, if a KBS can violate this IC, it is likely that there is a knowledge gap in the KB, that is, the KBS needs more rules.

### 2.2 Specifying Interaction with the Environment and other Agents

Nowadays, different notations can be employed to specify communication protocols: AUML interaction diagrams[1] or state machines as in (10). For the purpose of the proposed method, state machines are more suitable as the checking of the scenarios w.r.t. the protocols must be automated. Hence, a state machine view for the verified agent must also be supplied as an input to our method. Each state transition of the state machine owns a label that describes how the messages/stimuli that fire the transition are. This label is expressed in terms of message/stimulus schemas.

In addition to the state machine, a correspondence between message/stimulus schemas and acquired literals must be supplied. If a message/stimulus schema corresponds to a set of acquired literals $\{l_i\}_{i=1,..,n}$, any message/stimulus that matches that schema contains a model for the formula $\exists x_1 \exists x_2 ... \exists x_n (\bigwedge_{i=1,...,n} l_i)$ where $x_1, x_2, ..., x_n$ are all the free variables in $\bigwedge_{i=1,...,n} l_i$. This latter formula can be also viewed as a query.

### 2.3 Assumed Non-Monotonic Reasoning

CCR-2 rules can introduce new facts in the agent's state, but they can also delete already existing facts. This provides the agent's designer with the capability of building agents with non-monotonic reasoning. So, we could find production

[1] http://www.auml.org/

rules of the form $p \rightarrow Del(p)$ under CWA. This kind of rules (when $p$ is assumed to be provided) are not admissible in a RB from the point of view of classical logic or default logic (11), since they are logical inconsistencies. However, if we examine these rules from the point of view of temporal logic (12), and we rewrite them as $\neg p$ *atnext* $p$ (where the intended meaning for the operator *atnext* is: $\neg p$ holds at the next time point that $p$ holds), then these rules should be perfectly admissible in a RB. From our perspective, production rules should be interpreted as rules of the form $\neg p$ *atnext* $p$. If we admit rules of the form $\neg p$ *atnext* $p$, we situate ourselves quite far from the concept of inconsistency as defined in other works, so we are going to clarify the meaning of inconsistency in this work:

A deductive tree $T$ that deduces a pair of facts $F$ and $F'$ is *consistent* iff:

(a) $T$ does not contain a set of contradictory static facts, or
(b) the deductive subtree of $T$ that deduces $F$ does not deduce $F'$ in the end, and vice versa.

This definition implies that the deductive subtree that deduces a fact $F$ must not deny the other fact $F'$ that must hold at the same time than $F$, and vice versa.

When the agent executes a reasoning process, a deductive tree is evaluated and a sequence of rules is fired. A deductive tree defines a partial order for rule firings, so many sequences correspond to a certain deductive tree. The definition showed above is not more than a structural property to be fulfilled by the deductive trees built by the agent that we want to verify using our method. We will call this property $Tree\_Consistency(dt)$ where $dt$ is a deductive tree that is a *tree of rule firings* defined recursively by means of the constructor *tree* and the constant $NIL\_TREE$ (empty tree). As our method will simulate the agent's reasoning, it will discard any deductive process that implies the creation of an invalid deductive tree. Next, we will define this property formally:

$$Tree\_Consistency(dt) \equiv Tree\_Consistency\_Aux1(Boundary(dt)) \\ \wedge\ Tree\_Consistency\_Aux2(dt, \emptyset)$$

$$Tree\_Consistency\_Aux1(B) \equiv \\ \neg(\exists is \in INCONSISTENT\_SETS\ is \subset \bigcup_{r \in B} Assumed\_Facts(r))$$

$$Tree\_Consistency\_Aux2(dt, scope) \equiv (dt = NIL\_TREE) \vee \\ \exists r \exists a_1, \exists a_2 ... \exists a_n (dt = tree(r, (a_1, a_2, ..., a_n)), \\ scope\_in\_rule = scope \setminus Deduced\_Facts(r), \\ \neg((\exists f \in scope\_in\_rule, \exists f' \in Assumed\_Facts(r), (f = \neg f')) \vee \\ (\exists f \in Deduced\_Facts(r), \exists f' \in scope, (f = \neg f'))), \\ Tree\_Consistency\_Aux2(a_1, scope\_in\_rule \cup Assumed\_Facts(r)), \\ Tree\_Consistency\_Aux2(a_2, scope\_in\_rule \cup Assumed\_Facts(r)), \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ Tree\_Consistency\_Aux2(a_n, scope\_in\_rule \cup Assumed\_Facts(r)))$$

where $INCONSISTENT\_SETS$ is the set of the different inconsistencies to be considered, the function $Boundary(dt)$ returns the set of rule firings

that are leaves of the tree $dt$, the function $Deduced\_Facts(r)$ returns the facts deduced by the rule firing $r$ and the function $Assumed\_Facts(r)$ returns the static facts that must hold to permit the rule firing $r$.

In the definition above, the property $Tree\_Consistency\_Aux1$ specifies the condition (1) in the definition of consistent deductive tree above, and the property $Tree\_Consistency\_Aux2$ specifies the condition (2).

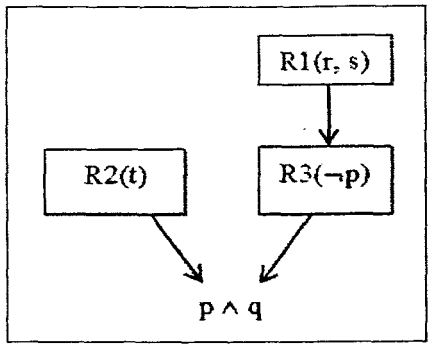


Figure 1: Example of an invalid deductive tree

Lets see an example of an inconsistent RB. Lets take the production rules $R1$: $r, s \rightarrow Del(p)$; $R2$: $t \rightarrow Add(p)$; $R3$: $\neg p \rightarrow Add(q)$ under CWA. In the figure 1 we can see the deductive tree for the conjunction $p \wedge q$ that is supposed to be the antecedent of another rule. The facts $p$ and $q$ are deducible and all the other facts are non-deducible. Obviously (see rule R3), in order to deduce $q$, $\neg p$ must be deduced beforehand, and after having deduced $\neg p$ it is not possible to deduce $p$. This example deserves an additional comment. If we assume that the rules are executed with forward chaining and we fire them in the sequence [R1, R3, R2] then the facts $p$ and $q$ will be both true in the final FB. However, if the rules are fired in the following sequence [R2, R1, R3] then the facts $\neg p$ and $q$ will be present in the final FB. With the first sequence, the fact $q$ was deduced first, and then the fact $p$; with the second sequence the facts were deduced the other way round. Our definition of inconsistency includes situations like this one, when the truth values of the goal facts depend on the order in which they are deduced.

Lets see an example of a RB that is consistent according to our definition, but inconsistent according to other definitions. Lets take the production rules $R1$: $n, u \rightarrow Add(q)$; $R2$: $s, \neg q \rightarrow Add(q)$; $R3$: $q, m, t \rightarrow Del(p)$; $R4$: $v \rightarrow Del(q)$ under CWA. In the figure 2 we can see the deductive tree for the conjunction $\neg p \wedge q$ that is supposed to be the antecedent of another rule. We want to deduce the $\neg p$ and $q$, and all the other facts are non-deducible. We can see that there are six different sequences of rules that correspond to the deductive tree of the figure 2. However, among them, only three sequences are feasible ([R4, R2, R1, R3], [R1, R3, R4, R2] and [R1, R4, R2, R3]), and all of these three sequences deduce the same truth values for $p$ and $q$.

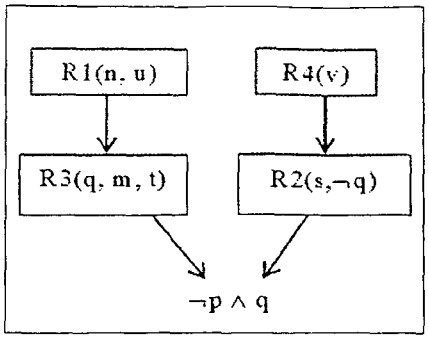

Figure 2: Example of a valid deductive tree

According to the above definition of inconsistency, it is clear that MECORI will not be able to verify some non-monotonic KBS. In particular, all the KBS whose deductive trees do not follow the *consistency* definition exposed above, for instance, the planners of STRIPS type.

## 3 Requirements for Getting an Inconsistency: Scenario

The aim of the proposed method, as it was explained in the section 1, will be to compute scenarios for an inconsistency described by an IC. Each scenario is formed by a description of the initial agent's state, a deductive tree of rule firings, and a partially ordered set of messages and/or stimuli. The proposed method will construct an object called *subcontext* to specify how the initial agent's state must be and which deductive tree must be executed in order to yield an inconsistency. There may be different initial agent's states and different deductive trees that lead to the same inconsistency. All the different ways to violate a certain IC will be specified by means of an object called *context*. Thus, a context will be composed of $n$ subcontexts. In turn, a subcontext is defined as a pair *(environment, deductive tree)* where an environment is made up of a set of *metaobjects*, and a deductive tree is a tree of rule firings.

A metaobject describes the characteristics that one object which can be present in the agent's state should have. For each type of CCR-2 object there will be a different type of metaobject: metaproposition, metaframe, metarelationship, metaattribute and metaid-attribute. In order to describe a CCR-2 object, a metaobject must include a set of constraints on the characteristics of the CCR-2 object. Some CCR-2 objects may include references to other CCR-2 objects (for example, a frame instance can have references to attributes and a relationship can include tuples of references to frame instances), so the counterpart metaobjects will contain references to other metaobjects. In the table below, the attributes of each type of metaobject are shown. The value of these attributes will represent the constraints described by each metaobject.

| CCR-2 Object | Metaobject | Attributes of the Metaobject |
|---|---|---|
| Frame | Metaframe | (identifier, is_restricted_exist, instance_of, subclass_of, metaattributes, metarelationships) |
| Attribute | Metaattribute | (identifier, is_restricted_exist, metaframe, metaid-attributes, value_conditions, cf_conditions) |
| Id-Attribute | Metaid-attribute | (identifier, is_restricted_exist) |
| Relationship | Metarelationship | (identifier, is_restricted_exist, type, tuples, conditions_for_each_tuple) |
| Proposition | Metaproposition | (identifier, is_restricted_exist, type, truth_value, conditions) |

Given that certain constraints expressed as arithmetic inequations can affect the attribute values and the certainty factors associated with CCR-2 objects, a different kind of metaobject called *condition* will represent them. Conditions will also appear in environments, together with metaobjects, and they will be referenced from and contain references to the metaobjects that participate in them. Considering the references among metaobjects and conditions, there can be one or more networks of metaobjects and conditions in one environment. Figure 3 illustrates an example of an environment describing a FB in which the formula $\neg Has(X, Water) \wedge X.temperature \geqslant 80$ is true, where the variable $X$ is declared as an instance of the frame *Car*. If there exists a CCR-2 object in the FB, for each metaobject in the environment, that satisfies all the requirements imposed on it, then the given formula will hold in the FB.

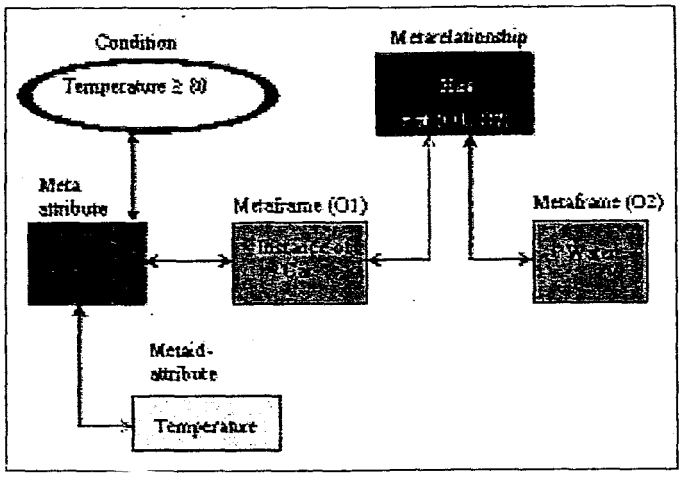


Figure 3: Environment

### 3.1 Temporal labels and constraints

A *goal* $h$ is a pair $(l, A)$ where $l$ is a literal and $A$ is a set of metaobjects associated with the object names and variables in $l$, that specifies the FBs in which the lit-

eral $l$ is satisfied. Moreover, a goal $(l, A)$ is static/dynamic/deducible/acquired/innate iff the literal $l$ is static/dynamic/deducible/acquired/innate.

For the purpose of executing a deductive tree, it may be required that a dynamic acquired fact $f$ holds in a rule, and later on, that the fact $\neg f$ holds in another rule. This situation may yield an apparently contradictory environment. To determine if it is a real contradiction, *temporal labels* will be associated with some constraints included in the goals $(l, A)$ and $(l', A')$ that entail $f$ and $\neg f$ respectively, to represent that these constraints must be satisfied in different rule firings (or moments). Each temporal label associated with a constraint identifies the rule firing where the constraint must be satisfied, and specifies that the constraint comes from a dynamic acquired fact. From these labels, the method will specify, as part of the resulting scenario, that a message/stimulus that matches schema $M$ and allows literal $l$ to hold must be received before a message/stimulus that matches schema $M'$ and allows literal $l'$ to hold is received, formally $M < M'$. *Temporal constraints*, like the one stated in the previous sentence, will define a partially ordered set of messages and/or stimuli schemas, in which the relationship $<$ expresses temporal precedence.

For each static acquired literal included in the KB, it will be required to produce a temporal constraint to establish that the message/stimulus (according to a schema) allowing the static acquired literal to hold must be received before the end of the deductive process. Consequently, to permit the proposed method to obtain the proper temporal constraints later, some temporal labels must also be associated with the constraints derived from static acquired literals. Besides, these labels must specify that the constraints have been obtained from a static acquired fact.

Moreover, the method has to generate temporal constraints to establish that some messages/stimuli allowing static acquired literals to hold must be received before the message/stimulus that allows a certain dynamic acquired literal to hold. Lets see the conditions in which these temporal constraints must be generated. Let $(R1, R2, ..., RN)$ be the sequence of rules that are fired as a result of evaluating a deductive tree according to the control mechanisms. Let $Ri$ *s.t.* $1 \leqslant i < N$ be a rule whose antecedent requires the dynamic acquired literal $Ld$ to hold, and let $M$ be a message/stimulus schema that entails $Ld$; let $Rj$ *s.t.* $i < j \leqslant N$ be a rule whose antecedent requires the dynamic acquired literal $\neg Ld$ to hold, and let $M'$ be a message/stimulus schema that entails $\neg Ld$. Then, it is clear that any message/stimulus schema $M1$ that entails a static acquired literal $Ls$ belonging to the antecedent of a rule $Rk$ *s.t.* $i \leqslant k < j$ must satisfy $M1 < M'$. The rationale for generating these temporal constraints will become clearer in the section 5 when an example is shown.

## 4 Description of the method

Computing the scenarios associated with an IC requires three steps:

1. Computing the context associated with the IC without taking into account the control mechanisms, and considering all the rules to form a unique group.

2. Computing the scenarios from the context associated with the IC and the control mechanisms.
3. Discarding invalid scenarios w.r.t. the communication protocols.

### 4.1 Computing the Context associated with the IC

Basically, the first step can be divided into two phases. In the first phase, the AND/OR decision tree associated with the IC is expanded following a backward chaining simulation of the real rule firings. The leaves of this tree are rules that only contain acquired facts in their antecedents. At this point, the difference between a deductive tree and a AND/OR decision tree should be explained. While a deductive tree can be viewed as one way and only one way for achieving a certain goal (that is, for deducing a bound formula or for firing a rule), an AND/OR decision tree comprises one or more deductive trees, therefore it specifies one or more ways to achieve a certain goal. During the first phase, metaobjects are built and propagated from a rule to another one. In this propagation, some constraints are added to the metaobjects due to the rule literals and the declaration part of the rules/IC, and some constraints are removed from the metaobjects due to the rule actions. In addition to the metaobjects, a set of assumed propositions and tuples (SAPT) are propagated and updated.

In the second phase, the AND/OR decision tree is contracted by means of context operations, and metaobjects associated with non-deducible facts and conditions associated with inequations are inserted in the subcontexts. Lets define the the following contexts operations: creation of a context, concatenation of a pair of contexts and combination of a list of contexts.

**Contexts Operation**

a) *Creation*: a context with an unique subcontext is created from a non-deducible goal $g = (l, A)$ and a rule $r$: $C(g, r) = \{(E, NIL\_TREE)\}$ where the environment $E$ comprises all the metaobjects included in $g$. The rule $r$ must be a rule that comprises the literal $l$ in its antecedent. If the literal $l$ is not innate (so it is related to a message/stimulus), some constraints of the metaobjects must be labelled with a temporal label indicating that these constraints must be satisfied at least in the firing of the rule $r$; in particular, constraints that state the truth value of a metaproposition, and constraints that state the truth value of a tuple in a metarelationship. The literal $l$ will hold in any agent's state that satisfies all the constraints specified in $E$.

b) *Concatenation of a pair of contexts*: let $C_1$ and $C_2$ be a pair of contexts and $Conc(C_1, C_2)$ be the context resulting from the concatenation, then: $Conc(C_1, C_2) = C_1 \cup C_2$.

c) *Combination of a list of contexts*: Let $C_1, C_2, ..., C_n$ be the list of contexts, and $Comb(C_1, C_2, ..., C_n)$ be the context resulting from the combination. The form of this resulting context is: $Comb(C_1, C_2, ..., C_n) = \{(E_{k1} \cup E_{k2} ... \cup E_{kn}, DT_{k1} * DT_{k2} ... * DT_{kn})$ *s.t.* $(E_i, DT_i) \in C_i\}$

c.1) *Union of environments* $(E_i \cup E_j)$: this operation consists of the union of the sets of metaobjects $E_i$ and $E_j$. After the union of two sets, it is necessary

to check whether any pair of metaobjects can be merged. A pair of metaobjects will be merged if they contain a pair of constraints $c_1$ and $c_2$ respectively such that $c_1$ and $c_2$ specify the same name. As a result of this fusion, the new metaobject could be invalid if it contains contradictory constraints not coming from dynamic acquired facts. In this case, the resulting environment will be invalid, and it will be discarded. Finally, if the resulting environment represents an invalid initial agent state, then this environment will also be discarded. Moreover, after the union of two environments, it is also necessary to check whether the resulting set of conditions can be satisfied or, in others words, whether the resulting set of conditions is feasible.

c.2) *Combination of deductive trees* ($DT_i * DT_j$): let $DT_i$ and $DT_j$ be deductive trees, then $DT_i * DT_j$ is the deductive tree that results from constructing a new tree whose root node represents an empty rule firing, and whose two subtrees are $DT_i$ and $DT_j$.

Basically, the creation operation is employed to work out the context associated with a non-deducible goal; the combination operation is employed to work out the context associated with a conjunction of literals from the contexts associated with the literals; and the concatenation operation is employed to work out the context associated with a disjunction from the contexts associated with the formulas involved in the disjunction.

These two phases are explained in detail in (13). However, there are some differences between the current step and the process explained in (13). These differences are related mainly to the context operations and the treatment of acquired facts and deductive trees. In (13) is explained a method for verifying an isolated KB System, so acquired facts are not considered, and the KB System is assumed to deal only with innate knowledge (external facts in (13)), and deduced knowledge.

### 4.2 Computing the Scenarios

In the second step of the method, a different scenario is derived from each subcontext in the context associated with the IC by adding a partially ordered set of messages and/or stimuli to the subcontext. In this step, some subcontexts may be discarded if they are impossible w.r.t. the control mechanisms. The partial order on the message/stimulus schemas reflects the temporal constraints derived from the control mechanisms and the deductive tree. These temporal constraints are generated as it was explained in the section 3. It may happen that more than one message/stimulus schema entails the same literal, so this aspect must be taken into account in building the temporal constraints to be added to the partially ordered set.

### 4.3 Discarding invalid Scenarios w.r.t. the Communication Protocols

In the previous steps, some scenarios have been computed for an IC. However, it may happen that some scenario obtained in the previous step describes im-

possible sequences of messages or stimuli w.r.t. the communication protocols. In order to check this, at least one path that satisfies all the temporal constraints must be found in the state machine. The first state of this path must be the state in which the agent begins its reasoning process.

## 5 Example of application

In this section we will show how the method can be applied to a small example. We will assume a deliberative agent that executes the sequence of rules that appears in the figure 4. For the sake of clarity and conciseness, the rules and the IC of this example are not represented in the CCR-2 format, and all the facts are propositional. In this example, the facts $q$ and $\neg q$ are dynamic acquired facts entailed by the messages $M$ and $M'$ respectively, whereas the fact $\neg r$ is a static acquired fact entailed by the stimulus $S$. Moreover, the fact $s$ belongs to the agent's innate knowledge, and the facts $t$ and $p$ are deducible.

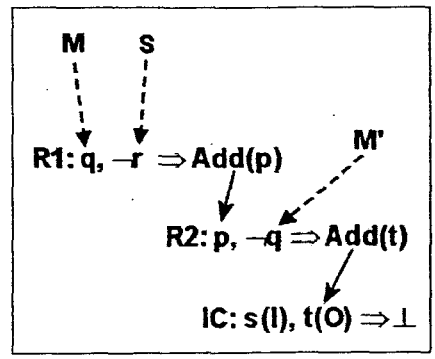


Figure 4: Example with an IC and two rules

The method begins expanding the AND/OR decision tree. First of all, it is necessary to bind each variable of the IC and each referenced object to a metaobject. Some constraints are derived from each IC literal, and they are added to the metaobjects (in this case, metapropositions). The resulting metapropositions are:

$$PROP1 = (id \rightarrow s, truth\_value \rightarrow true)$$
$$PROP2 = (id \rightarrow t, truth\_value \rightarrow true)$$

In addition to the metaobjects, the SAPT is created. This set contains the names of the propositions included in the IC and the tuples of relationships whose names appear in the IC that are not associated with dynamic acquired facts. So, initially, SAPT $= \{s, t\}$, because neither the fact $s$ nor the fact $t$ are dynamic acquired facts. The aim of the SAPT is to warrant the consistency of the non-monotonic reasoning in the sense explained in the section 2, concretely the second point of the consistency definition in section 2. The SAPT plays the

role of the *scope* parameter in the definition of the *Tree_Consistency* property. Unluckily, the metaobjects alone cannot warrant the consistency in all the cases. For example, if the SAPT is not used in the example of the section 2 (see figure 1), the inconsistency would not be detected in the simulation of the agent's execution, and that deductive tree would not be discarded.

Obtaining the context of an IC implies obtaining the context associated with each literal included in the IC. If it corresponds to a non-deducible goal, its context is created (see Creation Operation in section 4.1). In order to compute the context of a deducible goal, the method has to generate the contexts associated with all the rules that deduce the goal (conflict set), and then it has to concatenate them (in the contraction phase). To decide whether a rule deduces a goal, it is needed to check whether there exists any action in the rule that is unifiable with the goal. In the example of the figure 4, the IC comprises an innate literal (input literal) and a deducible literal (output literal). So, the method finds a rule (R2) to deduce the deducible literal.

In general, a CCR-2 rule premise contains a list of conjunctions joined by disjunction operators. Hence, to compute the context of a rule it is needed to calculate the context of each conjunction, and then they have to be concatenated (in the contraction phase). In order to compute the context of a conjunction, it is required to compute the context of each literal included in the conjunction. A pre-processing similar to that of an IC is performed over each conjunction before computing the contexts of the included literals. As a result of this, new metaobjects and conditions appear and some constraints are added to the metaobjects. In the rule R2, the metapropositions *PROP3(p)* and *PROP4(¬q)* are created.

The rule R2 contains only one conjunction with two literals $p$ and $\neg q$. While $p$ is a deducible fact, $\neg q$ is a dynamic acquired fact. In this example, the rule R1 can be employed to deduce the fact $p$. In the rule R1, the metapropositions *PROP5(q)* and *PROP6(¬r)* are created.

The SAPT propagated from the IC is updated while processing the rule R2, so now SAPT $= \{s, p\}$, since $t$ is deleted by the action of the rule R1, and $\neg q$ is a dynamic acquired fact. If the antecedent of the rule R2 had comprised the fact $\neg s$, a conflict would have been detected when updating the SAPT, and the rule R2 would have been discarded. Finally, the SAPT in the rule R1 is SAPT $= \{s\}$.

Once the AND/OR decision tree has been expanded completely, the tree is contracted by using the context operations, and the constraints generated for the non-deducible goals (inside the metaobjects) are propagated forward from the leaves of the AND/OR decision tree to the IC. Thus, all these constraints are collected in the context associated with the IC. In the example, the contexts associated with the non-deducible facts $s$, $q$, $\neg r$ and $\neg q$ are created, and next, the necessary combination operations are carried out until the context associated with the IC is computed. Every time a context is obtained from a combination operation in a rule R, this rule R is added to each deductive tree of the context as the new root node.

It is worth mentioning that while computing $Comb(C(p), C(\neg q))$ in the rule R2, an apparent conflict is detected between the metapropositions *PROP4* and *PROP5*, as they require different truth values for the same proposition $q$. However, there is no contradiction, since the facts $q$ and $\neg q$ are dynamic acquired facts, that is, the contradictory facts may hold in different moments. Hence, these metapropositions are merged, and the new metaproposition *PROP7* is yielded:

$$PROP7 = (id \rightarrow q, truth\_value \rightarrow \{true(R1, dynamic), false(R2, dynamic)\})$$

After applying the first step of the method, the resulting context associated with the IC is: *C(IC)= {({PROP1, PROP6, PROP7}, tree(R1, [tree(R2,nil)]) )}*, where these metapropositions are defined as:

$$PROP7 = (id \rightarrow q, truth\_value \rightarrow \{true(R1, dynamic), false(R2, dynamic)\})$$
$$PROP6 = (id \rightarrow r, truth\_value \rightarrow false(R1, static))$$
$$PROP1 = (id \rightarrow s, truth\_value \rightarrow true)$$

Next, in the second step, according to the control mechanism, it is determined that this deductive tree is evaluated by firing the sequence of rules $[R1, R2]$. Taking this into account, the following temporal constraints are derived from the metapropositions: $M < M'$, because the message $M$ must be received before the message $M'$, in order to allow the fact $q$ to hold first, and then to allow the fact $\neg q$ to hold later; and $S < M'$, because the stimulus $S$ must be received before the message $M'$, since, otherwise, the rule $R1$ will not be able to be fired before the rule $R2$. Thus, the partially ordered set is $\{M < M', S < M'\}$, and the scenario is $(C(IC), \{M < M', S < M'\})$

Finally, in the third step, the scenario is checked w.r.t. the agent's state machine, which describes the agent behaviour. We can see a fragment of this state machine in the figure 5. The reasoning process is supposed to begin in the state $q0$. It is clear that there is a path that satisfies all the temporal constraints imposed in the scenario, so the scenario is consistent with the state machine.

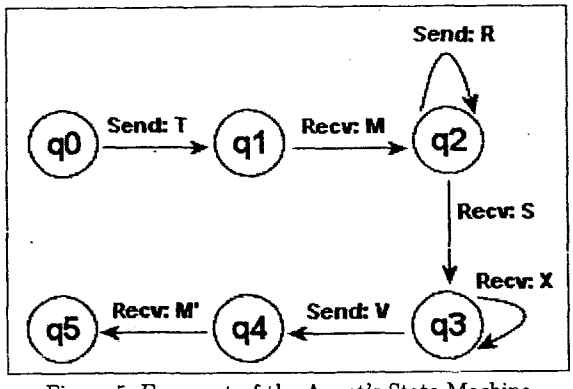


Figure 5: Fragment of the Agent's State Machine

## 6 Conclusion

In this paper, a formal method to verify the consistency of the reasoning process of a deliberative agent w.r.t. communication protocols has been presented. To the best of our knowledge, there is no other method or tool that also addresses this kind of verification. It is also noteworthy that the agent to be verified encompasses a hybrid KB that permits the representation of non-monotonic reasoning and arithmetic constraints.

## 7 Future Work

Mainly, there are two aspects of the proposed method that we want to improve: first, the validation of the deductive tree w.r.t. control mechanisms, more concretely, w.r.t. metarules; and second, the deletion of redundancy in the sets of temporal constrains by taking into account transitive dependencies and other aspects.

Moreover, we are working on the adaptation of the proposed method so that it can be applied to verify agents whose knowledge domain is expressed in a wide known ontology like OWL[2].

[2] http://www.w3.org/TR/owl-features/

# Bibliography


[1] de Kleer, J.: An assumption based TMS. Artificial Intelligence **28** (1986) 127–162

[2] Rousset, M.: On the consistency of knowledge bases: The COVADIS system, Proceedings ECAI-88, Munich, Alemania (1988) pp. 79-84.

[3] Ginsberg, A.: Knowledge-base reduction: A new approach to checking knowledge bases for inconsistency and redundancy, Proceedings of the AAAI-88 (1988) pp. 585-589.

[4] de Antonio, A.: Sistema para la verificación estructural y detección de inconsistencias en bases de conocimientos. Final year project, Facultad de Informática, UPM (1990)

[5] Meseguer, P.: Incremental verification of rule-based expert systems, Proceedings of the 10th. European Conference on AI (ECAI'92) (1992) pp. 840-844.

[6] Dahl, M., Williamson, K.: A verification strategy for long-term maintenance of large rule-based systems, Workshop Notes of the AAAI92 WorkShop on Verification and Validation of expert Systems (1992) pp. 66-71.

[7] Ayel, M.: Protocols for consistency checking in expert system knowledge bases, Proceedings of the European Conference on Artificial Intelligence (ECAI'88) (1988) pp. 220-225.

[8] Ayel, M., Laurent, J.P.: Validation, Verification and Test of Knowledge-Based Systems: SACCO-SYCOJET: Two Different Ways of Verifying Knowledged-Based Systems. John Wiley publishers (1991)

[9] de Antonio, A., Cardeñosa, J., Martínez, L.: GKR: A generic model of knowledge representation. Volume II, Student Abstracts., Proceedings of the 12th National Conference on Artificial Intelligence (AAAI94) (1994) pp. 1438.

[10] d'Inverno, M., Kinny, D., Luck, M.: Interaction protocols in agentis, Third International Conference on Multi-Agent Systems (ICMAS98) (1998) 261–268

[11] Antoniou, G.: Verification and correctness issues for nonmonotonic knowledge bases. International Journal of Intelligent Systems **12** (1997) 725–738

[12] Kröger, F.: Temporal Logic of Programs. Springer-Verlag (1987)

[13] Ramírez, J., de Antonio, A.: Knowledge base semantic verification based on contexts propagation, Notes of the AAAI-01 Symposium on Model-based Validation of Intelligence (2001)