Consistency Verification of a Non-monotonic Deductive System based on OWL Lite

Jaime Ramírez and Angélica de Antonio

Technical University of Madrid
Madrid, Spain

Abstract. The aim of this paper is to show a method that is able to detect a particular class of semantic inconsistencies in a deductive system (DS). A DS verified by this method contains a set of production rules, and an OWL Lite ontology that defines the problem domain. The antecedent of a rule is a formula in Disjunctive Normal Form, which encompasses first-order literals and linear arithmetic constraints, and the consequent is a list of actions that can add or delete assertions in a non-monotonic manner. By building an ATMS-like theory the method is able to give a specification of all the initial Fact Bases (FBs), and the rules that would have to be executed from these initial FBs to produce an inconsistency.

1 Introduction

The purpose of this paper is to present a method for verifying the semantic consistency of deductive systems (DSs) that deal with production rules and an OWL Lite ontology[1]. One of the most interesting facets of the proposed method is that it is able to deal with non monotonic reasoning. As the consequent of the production rules is allowed to add and delete facts about individuals in the Fact Base (FB), the behaviour of the verified DS is *non-monotonic*.

Some methods or tools intended to verify the consistency of a DS (mostly rule based systems) build a model of the DS (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 follow another approach in which the starting point is one of the inconsistencies that might be possibly deduced by the verified DS, and the goal is to compute a description of the initial FBs in which the DS 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].

Section 2 of this paper mentions other methods that have also been proposed to verify the consistency of a hybrid DS or non-monotonic DS. In section 3, some aspects are explained related to the DSs to be verified. Section 4 describes how to specify the inconsistencies that are verified by this method. Section 5 explains how this method

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

Ramírez J. and de Antonio A. (2005).
Consistency Verification of a Non-monotonic Deductive System based on OWL Lite.
In *Proceedings of the 3rd International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems*, pages 19-28
DOI: 10.5220/0002573400190028
Copyright © SciTePress

specifies the way in which a DS can deduce an inconsistency, if possible. In section 6, the procedure for detecting an inconsistency is outlined and illustrated by means of a simple example. We end with some conclusions and future works derived from our work.

2 Previous Work

There are not many methods that are able to deal with the verification of non-monotonic DSs based on rules. Among them, we can highlight Antoniou's method [4], which analizes DSs expressed in default logic, and Wu & Lee's method [5], which tests DSs with production rules under the closed world assumption. While Antoniou's method is inspired on tabular methods for monotonic DSs, Wu & Lee's method models the DS as a high level extended Petri net. Analogously, only a few methods have been proposed that are able to verify hybrid systems, that is, systems that combine two different knowledge representation formalisms, one for representing the problem domain, and another for representing the inference knowledge. The first work that dealt with hybrid systems was that of Lee & O'Keefe [6], which characterized a set of new types of anomalies that appear as a result of considering the subsumption relationship among literals in different rules. In order to detect those anomalies, Lee & O'Keefe presented a method with a tabular approach. Finally, we will mention another method to verify hybrid systems, which was proposed by Levy and Rousset [7], and is the most advanced method proposed to date to test hybrid systems. This method can be applied to hybrid systems expressed in the CARIN language [8], which is a language that combines the flexibility of the Horn clauses with the expressiveness of the $\mathcal{ALNCR}$ Description Logic (DL). The method provided is grounded in theoretical results related to the query containment problem, studied in the database literature. However, the main drawback of this method is that it is required to evaluate the same query for each valid input, and this process may imply a very high computational cost. Thus, to the best of our knowledge, no method, except for ours, can deal with hybrid DSs with non monotonic reasoning. Moreover, our method does not need to check the DS w.r.t. each possible input, since it begins from the inconsistency, and then it tries to build the description of a conflictive input working backwards.

3 Characteristics of the Deductive System to be Verified

The DS is made up of a Knowledge Base (KB) and an inference engine. In turn, the DS's KB consists of an OWL Lite ontology and a set of production rules. Let us describe each part in more detail:

3.1 OWL Lite ontology

OWL language was intended to associate meaning to the web contents. In addition, OWL allows for defining a shared vocabulary that several agents (which may be endowed with DSs) can process and utilize to exchange information. OWL is a layered

language because it specifies a hierarchy of three sub-languages. They are, sorted by decreasing degree of expressiveness, OWL Full, OWL DL and OWL Lite. Although OWL Full and OWL DL are more expressive than OWL Lite, the utilization of OWL Lite for reasoning purposes is more recommended, as long as the entailment problem for OWL Full is undecidable, and quite inefficient for OWL DL. An OWL Lite ontology consists of a set of axioms and a set of facts. The axioms define some class taxonomies where each class comprises a set of properties, whereas each fact defines an individual. Moreover, an OWL property is either a data-valued property or an object-valued property. In OWL Lite, the cardinality of the properties must be 0 or 1. Moreover, a property can be defined to be transitive, symmetrical or inverse of another property. The properties can be arranged to make up taxonomies of properties, for example, the property *HasFather* can be defined as a subproperty of the property *HasRelative*.

3.2 Production rules

The rule form considered by the method is: $(l_{11}, l_{12}, ..., l_{1w} \vee$ **,...,** $\vee\, l_{m1}, l_{m2}, ..., l_{ms}) \rightarrow a_1, a_2, ..., a_t$ where the antecedent part contains a disjunction of m conjunctions of literals (l_{ij}), and the consequent part contains a list of actions (a_k). A *literal* is an atom (or atomic formula), a negated atom (except when the atom is $Different(I1, I2)$, since it must not be negated) or a linear arithmetic constraint. The variables must be preceded by ?, and the types of the variables can be determined taking into account the OWL Lite axioms. The kinds of atoms that can occur in the antecedent part are: $SubClass(C1, C2)$, $Instance(I, C)$, $SubProperty(P1, P2)$, $Different(I1, I2)$ and $PROPERTY\ (I1, VALUE)$, where each argument written in capital letters can be a variable or an object name. Hence, $PROPERTY$ may be a variable representing any property. The intended meaning for the literal $\neg PROPERTY\ (I1, VALUE)$ is:
$\neg PROPERTY(I1, VALUE) \equiv \exists VALUE1(PROPERTY(I1, VALUE1)$
$\wedge Different(VALUE, VALUE1))$. This meaning is established taking into account that the maximum cardinality for the OWL Lite properties is 1. The linear arithmetic constraints are intended to represent complex relationships over some data-valued properties on the real domain. The syntax for these constraints is the same as the syntax specified for the DL reasoner RACER[2].

Basically, we admit actions in the consequent part to be addition actions or deletion actions. By means of an addition action, a rule can add a fact to the FB, whereas by means of a deletion action, a rule can remove a fact from the FB. Syntactically, an action can take the form $Add(Individual\_Atom)$ or $Del(Individual\_Atom)$ where $Individual\_Atom$ is either $Instance(I, C)$ or $PROPERTY(I1, Value)$.

3.3 Dynamic aspects

The DS is assumed to contain an inference engine able to execute the rules following a forward or backward chaining. Furthermore, the DS must support a DL reasoner able to deal with the entailment problem. As an OWL Lite ontology can be translated into the description logic $\mathcal{SHIF}$ [9], DL reasoners such as RACER can be used to instantiate

[2] http://www.cs.concordia.ca/ haarslev/racer/racer-manual-1-7-7.pdf

a literal of a rule on demand if possible. If a certain literal cannot be instantiated with the facts entailed by the ontology, and the DS follows a backward chaining, the rule inference engine will try to find a rule that deduces that literal.

When a rule is fired, we assume that all the actions belonging to the consequent of the rule are executed sequentially.

Facts in a DS are classified into two categories: a *deducible fact* is a fact that is obtained from the execution of the DS; and an *external fact* is a fact that cannot be deduced by the DS and can only be obtained from an external source.

3.4 Non-Monotonicity and Inconsistency

Rules can introduce new facts in the FB, but they can also delete already existing facts. This provides the DS's designer with the capability of building DSs with non-monotonic reasoning. Consequently, we could find production rules of the form $p \rightarrow Add(\neg p)$ under Open World Assumption (OWA). This kind of rules (when p is assumed to hold) are not admissible in a monotonic KB, since they are logical inconsistencies. If we admit rules of the form $p \rightarrow Add(\neg p)$, we situate ourselves quite far from the concept of inconsistency in monotonic DSs 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 conjunction of facts F and F' is *tree consistent* iff:
>
> 1. T does not contain a set of contradictory external facts, or
> 2. the deductive subtree of T that deduces F must not deduce $\neg F'$ in the end, and vice versa.

This definition is not more than an structural property to be fulfilled by the deductive trees built by the DS that we want to verify using the method. As the method will simulate the DS execution, it will discard any deductive process that implies the creation of an invalid deductive tree. Let us see an example of an inconsistent set of rules. For the sake of clarity, a simplified notation for the rules will be employed in this example. According to this notation, $\neg p$ denotes a fact that is contradictory with the fact p w.r.t. the ontology axioms. Let us take the production rules $R1$: $r, s \rightarrow Del(p), Add(\neg p)$; $R2$: $t \rightarrow Add(p)$; $R3$: $\neg p \rightarrow Add(q)$ under OWA. In the figure 1 we can see the deductive tree for a 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 external. 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.

4 Specification of Semantic Inconsistencies

Each semantic inconsistency that must be considered is represented by means of an Integrity Constraint (IC). The IC form is: $\exists x_1 \exists x_2 ... \exists x_n \ (l_1(scope_1) \wedge l_2(scope_2) \wedge ... \wedge l_k(scope_k)) \Rightarrow \bot$. A scope is associated with each literal to specify the kind of data referenced in the literal (input or output). A literal with input scope states something about the initial FB, while a literal with output scope states something about the final FB (resulting after an execution of the DS).

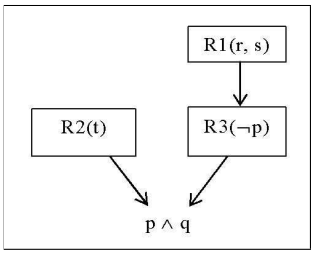

Fig. 1. Example of an invalid deductive tree

5 Specification of the Contexts

5.1 Describing Fact Bases

The proposed method will construct an object called *subcontext* to specify how the initial FB must be and which deductive tree must be executed in order to cause an inconsistency. There can be different initial FBs and different deductive trees that lead to the same inconsistency. An object called *context* will gather all the different ways to violate a given IC. Consequently, a context will be composed of *n* subcontexts. In turn, a subcontext is defined as a pair *(environment, deductive tree)* where an environment is composed of a set of *metaobjects*, and a deductive tree is a tree of rule firings. A metaobject describes characteristics that one object that can be present in the FB should have. For each type of OWL Lite object there will be a different type of metaobject. In order to describe an OWL Lite object, a metaobject must include a set of constraints on the characteristics of the OWL Lite object. All the metaobjects's attributes are outlined below:

Metaclass = (identifier, subclass_of, metaindividuals)
MetaObjectProperty = (identifier, pairs_of_metaindividuals, subproperty_of)
MetaDataTypeProperty = (identifier, pairs_of_metaindividual-value, subproperty_of)
MetaIndividual = (identifier, instance_of, objectproperties, datatypeproperties, differentfrom)
MetaValue =(conditions)

Given that certain constraints expressed as arithmetic inequations can restrict the datatype property values, a different kind of metaobject called *condition* will represent these constraints. The attributes of a condition are *expression* and *values*. Lets see an example of an environment describing a FB in which the formula $Instance(?X, Person) \wedge Feels(?X, Tiredness) \wedge Temperature(?X, ?temp) \wedge ?temp \geqslant 36.5$ holds. If there exists an OWL Lite 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. The environment is defined as {CLASS1, IND1, IND2, OPRO1, DTPRO1, VALUE1, COND1} where:

$CLASS1 = (Person, , \{IND1\})$
$IND1 = (, \{CLASS1\}, \{OPRO1\}, \{DTPRO\},)$
$IND2 = (Tiredness, \{OPRO1\}, ,)$
$OPRO1 = (Feels, \{(IND1, IND2)\},)$
$DTPRO1 = (Temperature, \{(IND1, VALUE1)\},)$
$VALUE1 = (\{COND1\})$
$COND1 = (\text{"}?temp \geqslant 36.5\text{"}, [VALUE1])$

CLASS1 is a metaclass, IND1 and IND2 are metaindividuals, OPRO1 is a metaObjectProperty, DTPRO1 is a metaDataTypeProperty, VALUE1 is a metaValue1, and finally COND1 is a condition. As defining the metaobjects, two consecutive commas represent an empty field.

5.2 Contexts Operations

We will define the following contexts operations: creation of a context, concatenation of a pair of contexts and combination of a list of contexts. These operations will be employed by the method to compute the scenarios, as we will see later on. Before defining the context operations, the goal object will be defined: A *goal* g 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 literal l is satisfied without using DL reasoning. Moreover, a goal (l, A) is external/deducible iff the literal l is external/deducible.
a) *Creation*: a context with a unique subcontext is created from an external goal $g = (l, A)$: $C(g) = \{(E, EMPTY\_TREE)\}$ where the environment *E* comprises all the metaobjects included in *A*.
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 $(c_1 \wedge c_2)$ entail that both metaobjects represent the same OWL Lite object. This will happen if both metaobjects should have the same value in the *identifier* attribute according to the ontology axioms. Finally, if the resulting environment represents an invalid initial FB, then this environment will also be discarded. This last check will be carried out with the help of the DL reasoner RACER.

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.

6 Computing the Context associated with an Integrity Constraint

The process to compute the context associated with an IC is divided into two steps. Next, these steps will be explained.

6.1 First Step

The first step can be considered as a pre-processing of the set of rules. In the second step, a backward chaining simulation of the real rule firings is carried out without making

calls to the DL reasoner (except for consistency checks in the union of environments, see 5.2, or in the updates of the set of assumed individuals, as we will see in the second step). However, in a real execution of the DS some literals in the rule antecedents may be instantiated thanks to these DL reasoner calls. In order to fill this gap in the simulation, some new rules derived from the ontology axioms and already existing rules are added to the set of rules. In particular, the generation of new rules is related to the presence of deduced literals in the rules. Sometimes, a rule adds a new fact to the FB that matches a literal in another rule's antecedent; in that case, we will say in the simulation that the first rule can be chained with the literal. In other cases, a rule adds a new fact to the FB that does not match directly a literal in a rule antecedent, but it actually does it indirectly, because the new fact allows the DL reasoner to deduce another fact that does match the literal. For the purpose of simulating properly this kind of inference situations, the set of rules must be pre-processed. Next, we will explain how the new rules are computed from the ontology axioms and already existing rules:

Deducing the literal $\neg Instance(ID1, C)$:
According to the syntactical restrictions explained in 3.2, no rules can deduce directly this kind of literals, but DS actually can indirectly deduce it these ways:

1. $R(ID1, ID2), \neg subclass(C, Domain(R)) \rightarrow \neg Instance(ID1, C)$
2. $R(ID2, ID1), \neg subclass(C, Range(R)) \rightarrow \neg Instance(ID1, C)$

Where R is any property. Thus, in each rule whose antecedent contains a conjunction c where the deducible literal $\neg Instance(ID1, C)$ occurs, the conjunction c will be replaced with the new conjunctions:
$Substitute(c, "\neg Instance(ID1, C)", "R(ID1, ID2), \neg subclass(C, Domain(R))")$ and,
$Substitute(c, "\neg Instance(ID1, C)", "R(ID2, ID1), \neg subclass(C, Range(R))")$
where the function $Substitute(c, s1, s2)$ returns the conjunction resulting from replacing the string $s1$ with the string $s2$ in the conjunction c.
Deducing the literal $Instance(ID, C)$:
Following an analogous reasoning to the previous replacement, in each rule whose antecedent contains a conjunction c where the deducible literal $Instance(ID1, C)$ occurs, two new conjunctions must be added:
$Substitute(c, "Instance(ID1, C)", "R(ID1, ID2), subclass(C, Domain(R))")$ and,
$Substitute(c, "Instance(ID1, C)", "R(ID2, ID1), subclass(C, Range(R))")$.

Furthermore, given that any individual a that is instance of a class A is also instance of any superclass of A, then another conjunction must be added:
$Substitute(c, "Instance(ID1, C)", "Instance(ID1, C1), subclass(C1, C)")$
Deducing transitive object properties:
If the object property R is defined to be transitive, then in each rule whose antecedent contains a conjunction c where the deducible literal $R(ID1, ID2)$ occurs, the new conjunction must be added:
$Substitute(c, "R(ID1, ID2)", "R(ID1, ?X), R(?X, ID2")$ st. the variable X does not occur in the conjunction c.
Deducing symmetric object properties:
If the object property R is defined to be symmetric, then in each rule whose antecedent contains a conjunction c where the deducible literal $R(ID1, ID2)$ occurs, the new con-

junction must be added:
$Substitute(c, "R(ID1, ID2)", "R(ID2, ID1)")$.
Deducing inverse object properties:
If the object property R^{-1} is defined to be inverse of the object property R, then in each rule whose antecedent contains a conjunction c where the deducible literal $R^{-1}(ID1, ID2)$ occurs, the new conjunction must be added:
$Substitute(c, "R^{-1}(ID1, ID2)", "R(ID2, ID1)")$ and vice versa.

6.2 Second Step

Basically, the second 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 external facts in their antecedents. At this point, the difference between a deductive tree and an 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 corresponding with each variable of a rule/IC that is being processed and each referenced OWL Lite name, and these metaobjects are propagated from a rule to another one. In this propagation, some constraints are added to the metaobjects due to the rule literals, and some constraints are removed from the metaobjects due to the rule actions, because any constraint deduced by an action is not required to be satisfied by the initial FB any more. In addition to the metaobjects, a set of assumed individuals (SAI) is propagated and updated. The aim of SAI is to warrant that the expanding deductive tree fulfills the second condition of the $Tree\ Consistency$ definition (see 3.4). The first condition of the $Tree\ Consistency$ definition is checked in the union of environments (see 5.2) during the next phase.

Figure 2 shows an example with a rule R1 and an IC, as well as the deductive tree expanded by the proposed method for this IC. In this example, the T property literals are deducible, whereas the rest of literals are external. This figure also shows the names of the metaobjects built for the variables and the OWL Lite names, as well as the two propagations of metaobjects through the two goal-action chainings. We will follow the trajectory of metaindividual I2 from the IC, where it is created for the variable $?X$, to the rule R1. In the IC, I2 is created as $(, , \{OPR1\}, ,)$ (see the format of the metaobjects in section 5.1). Then, the reference to $OPR1$ is removed from I2 in the first chaining, because the action deduces a pair of the object property T in which I2 is involved. Next, in the rule R2, I2 is required to appear in two pairs, one of the object property R, and another of the object property T; therefore I2 is updated to $I2 = (, , \{OPR1, OPR2\}, ,)$. Now, I2 is involved in another chaining, this time from R2 to R1, and in this chaining the reference to $OPR1$ is removed from I2 due to the simulation of the action effect. Finally, in the rule R1, a reference to the object property R and a constraint stating that the individual I2 is an instance of class A are added to I2. For the example of the figure 2, a SAI is created in the IC, so that $SAI = \{SubProperty\ (DPR1, DPR2), DPR1(I1, V1),\ V1 > 5, T(I2, I3),\ Different\ (I3, a)\}$. Then, in the first chaining the action removes $T(I2, I3)$ from SAI, and when SAI gets to

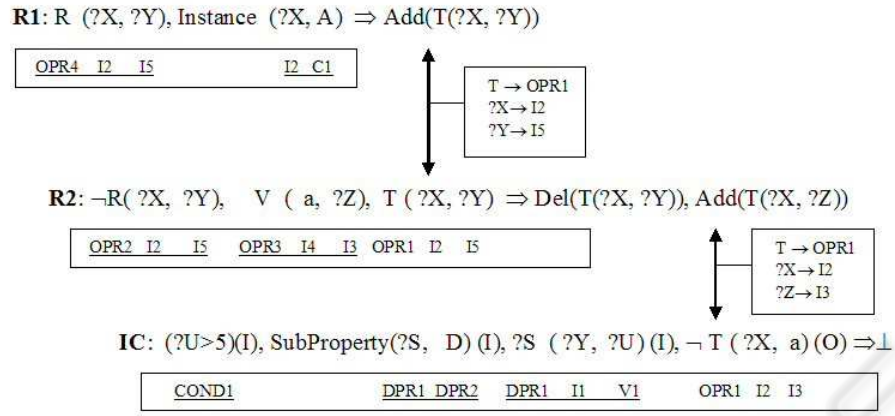

Fig. 2. Deductive tree of the case study

the conjunction of R2, it is updated so that $SAI = \{SubProperty\ (DPR1, DPR2), DPR1(I1, V1), V1 > 5, V(a, I3), T(I2, I5), Different(I3, a)\}$. Finally, in R1, $SAI = \{SubProperty(DPR1, DPR2), DPR1(I1, V1), V1 > 5, V(a, I3), Different(I3, a), Instance(I2, C1)\}$. As we can just see in this paragraph, the example of the figure 2 does not raise any inconsistency propagating SAI. However, if the external literal $V(a, a)(I)$ was added to IC, then in the antecedent of R2, SAI would be $\{SubProperty(DPR1, DPR2), DPR1(I1, V1), V1 > 5, V(a, I3), T(I2, I5), Different(I3, a), V(a, a)\}$, which is inconsistent because it forces the object property V to have two pairs $(a, I3)$ and (a, a) st. $Different\ (I3, a)$. If SAI turns to be inconsistent w.r.t. the ontology axioms, the $Tree\ Consistency$ property does not hold for the current deductive tree, and then the current rule must be discarded.

In the second phase, the AND/OR decision tree is contracted by means of context operations, so that metaobjects in external goals and conditions related to metaobjects in external goals are inserted in the subcontexts of the context associated with the IC. Basically, the creation operation is employed to work out the context associated with an external 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. Let us see the context associated with the IC in the example of the figure 2: $C(IC) = \{SUBC1\} = \{(\{C1, I1, I2, I3, I5, I5', I6, DPR1, DPR2, OPR3, OPR5\ , V1\ , COND1\}, tree(R1, [tree(R2, [EMPTY\_TREE])]))\}$ where:

$C1 = (A, , \{I2\})$
$I1 = (, , , \{DPR1\},)$
$I2 = (, \{A, R1)\}, \{OPR5\}, ,)$
$I3 = (, , \{OPR3\}, , \{I6\})$
$I5' = (, , , , \{I5\})$
$I5 = (, , \{OPR5\}, , \{I5'\})$
$I6 = (a, , \{OPR3\}, , \{I3\})$
$/* I6 = I4 + I3' */$
$DPR1 = (, \{(I1, V1)\}, \{DPR2\})$
$DPR2 = (D, ,)$
$OPR3 = (V, \{(I6, I3)\},)$
$OPR5 = (R, \{(I2, I5),$
$(I2, I5')\},)$
$/* OPR5 = OPR2 + OPR4 */$
$V1 = (\{COND1\})$
$COND1 = (''?U > 5'', \{V1\})$

The two phases of the second step are explained in detail for a frame-like knowledge representation formalism called CCR-2 in [10].

7 Conclusion and Future Work

In this paper, a formal method to verify the consistency of the reasoning process of a DS has been presented. It is noteworthy that the DS to be verified encompasses a KB, endowed with an OWL Lite ontology and a set of production rules, which permits the representation of non-monotonic reasoning and arithmetic constraints. So far, most of the efforts dedicated to the consistency verification of DSs have focused on the verification of a set of rules ignoring the domain knowledge. One of the few works that has dealt with the verification of both the set of rules and the domain knowledge was proposed in [7]. That work explains how to verify DSs whose domain knowledge is expressed in a rich language based on a DL. In our approach, however, we have chosen to sacrifice expressiveness (OWL Lite instead of OWL DL) in favour of efficiency, so that the proposed method can be applied to large systems; one of our next steps will be to show this empirically. We are currently studying more deeply if the proposed preprocessing rules in 6.1 is exhaustive, so as to ensure the completeness of the simulation in the second step (see 6.2). Besides, we are working on an extension of the proposed method that verifies the reasoning module of a deliberative agent cohabiting a dynamic environment with other agents. In this dynamic environment the truth value of some external facts may change during the reasoning process as a result of the reception of new messages or stimuli coming from the environment of the verified agent.

References

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. Antoniou, G.: Verification and correctness issues for nonmonotonic knowledge bases. International Journal of Intelligent Systems **12** (1997) 725–738
5. Wu, C.H., Lee, S.J.: Knowledge verification with an enhanced high-level petri-net model. IEEE Expert **12** (1997) 73–80
6. Lee, S., O'Keefe, R.M.: Subsumption anomalies in hybrid knowledge based systems. International Journal of Expert Systems **6** (1993) 299–320
7. Levy, A.Y., Rousset, M.: Verification of knowledge bases on containment checking. Artificial Intelligence **101** (1998) 227–250
8. Levy, A.Y., Rousset, M.: CARIN: A representation language combining horn rules and description logics, Proceedings ECAI'96 (1996)
9. Horrocks, I., Patel-Schneider, P.F.: Three theses of representation in the semantic web. In: Proc. of the Twelfth International World Wide Web Conference (WWW 2003), ACM (2003) 39–47
10. 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) `http://ase.arc.nasa.gov/mvi/abstracts/index.html`.