Checking Integrity Constraints in Deductive Systems based on Production Rules and a Description Logic Terminology

Jaime Ramírez and Angélica de Antonio

Technical University of Madrid
28660, 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 first order production rules, and a description logic terminology that defines the domain problem. By building an ATMS-like theory the method is able to give an 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 illustrate a method for verifying the semantic consistency of deductive systems (DS) that deal with production rules and a description logic (DL) terminology (also called TBox). In this sense, DSs that use a powerful language for modelling the reasoning process and a DL language for modelling general knowledge about the problem domain are often considered *hybrid* reasoning systems. Hybrid reasoning systems were popular in the 1980's; lately, the topic has regained attention, focusing on knowledge bases with a DL component for concept definitions and a logic-programming component for assertions about individuals [1] [2]. DLs are a family of languages that are especially proper to model complex constraints related to concepts structured in hierarchies. On the other hand, the production rules are more suitable to represent the behaviour of a system, sometimes non-monotonically, expressing contingent properties. DSs based on DLs have been employed successfully in configuration problems, and they constitute a promising area in the medicine and the database fields.

2 Scope of the method

Our method is able to verify a DS that is formed by a set of production rules and a Tbox. The TBox is a set of DL formulas whose form can be $C \sqsubseteq D$ or $C = D$ where C and D are concepts. Moreover, the production rule form is: $(x_1 \in T_1, x_2 \in T_2, ..., x_n \in T_n)$ $(l_{11}, l_{12}, ..., l_{1w} \vee\ l_{21}, l_{22}, ..., l_{2v} \vee, ..., \vee\ l_{m1}, l_{m2}, ..., l_{ms}) \rightarrow a_1, a_2, ..., a_t$ where the antecedent part contains a declaration part for the variables used in the rule (x_r),

and a disjunction of m conjunctions of literals (l_{ij}); and the consequent part contains a list of actions (a_k). Each variable used in the rule must be declared as an atomic concept defined in the TBox, so $x_r \in T_r$ is intended to mean that the variable x_r only can be bound to an individual a s. t. $T_r(a)$ holds. A *literal* is either a first-order logic atom or a negated atom. The predicate of an atom corresponds to a DL atomic concept or a DL atomic role, not necessarily defined in the TBox. An *assertion* or *fact* is a ground literal. Assertions in a DS are classified into two categories: a *deducible assertion* is an assertion that is obtained from the execution of the DS; and an *external assertion* is an assertion that cannot be deduced by the DS and can only be obtained from an external source. We will see the Fact Base (FB) as a set of assertions. We admit actions to be addition actions or deletion actions for assertions. DSs can use two different approaches for the management of the negation: closed world assumption or 3-valued logic. Each semantic inconsistency that must be considered is represented by means of an Integrity Constraint (IC). An IC defines a consistency criterion over input data, output data or input and output data. The IC form is: $\exists x_1 \in T_1 ... \exists x_n \in T_n$ $(l_1(scope_1) \wedge l_2(scope_2) \wedge ... \wedge l_k(scope_k)) \Rightarrow \perp$.
The DS is assumed to execute with forward chaining or backward chaining. Explicit control mechanisms and meta-rules are not considered by the proposed method. The DS can reason non-monotonically, even, the DS may employ rules of the form $p \rightarrow \neg p$, so we will situate ourselves quite far from the concept of logical inconsistency as defined in other works. Hence, we are going to clarify the meaning of logical inconsistency in this work: a *logical inconsistency* is reached when, in order to deduce a pair of facts F and F', it is necessary to assume the truth of a set of contradictory external facts, or when at any time in the process of deducing F (F'), the fact $\neg F'$ ($\neg F$) is deduced, and the fact F'(F) is not deduced later in the same deductive process. This notion of logical inconsistency is explained with more depth in [3].

3 Contexts

As the output, the method, for each IC that can be violated, must generate a report explaining the requirements that the initial FB must fulfill so that it is possible to execute, starting from the initial FB, a certain deductive path (sequence of rule firings) that causes the inconsistency described by the IC. The method will construct an object called *subcontext* to specify how the initial FB must be and which deductive path must be executed in order to cause an inconsistency. There can be different initial FBs and different deductive paths that lead to the same inconsistency. An object called *context* will specify 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 path)* where an environment is composed of a set of *metaobjects*, and a deductive path is a sequence of rule firings. A metaobject describes the characteristics that one object which can be present in a FB should have. We will distinguish three kinds of objects in a FB: individuals, concepts and roles. Hence, we will consider the following kinds of metaobjects: *Metaindividual(identifier, subinstance_of, roles)*, *Metaconcept(identifier, individuals)* and *Metarole(identifier, tuples)*. In order to describe a FB object, a metaobject must include a set of constraints on the characteristics of the FB object. These constraints will be included in the fields of the metaobjects. For instance, the $tuples$ field will comprise the tuples that must belong or not belong (they will appear as negated tuples) to the role described by the metarole. Each tuple, in turn, will include a list of metaindividual names. Lets see an example of an environment describing a FB in which the formula $Owns(X, umbrella) \wedge Person(X)$ is true. This environment is $\{$*IND1(*$\_$*,* $\{$*Person*$\}$*,* $\{$*Owns*$\}$*), IND2 (umbrella,*$\_$*,* $\{$*Owns*$\}$*), CON1(Person,* $\{$*IND1*$\}$*), ROLE1(Owns,* $\{$*(IND1, IND2)*$\}$*)*$\}$. If there exists an object in the FB, for each metaobject in the environment, that satisfies all the requirements imposed on it, then the given formula will be true in the FB.

4 Operation of the method

In order to analyze the consistency of a DS, our method has to compute the context associated to each IC. If this context results to be empty, that means that there is not any valid initial FB that leads to the violation of the IC. The method implements a backward chaining simulation of the real rule firing. The recursive calls finish when, in the process of computing the context associated with a fact, this fact is external. Basically, the method can be divided into two phases. In the first phase, the AND/OR deductive tree associated with the IC is expanded. The leaves of this tree are rules that only contain external facts. During the first phase, some metaobjects are associated to the object names and the variables in the rules, and they are propagated from a rule to another one, and updated in each rule. In this updating process some constraints derived from the literals and variable declarations are inserted into the metaobjects. Moreover, in this updating process, the satisfacibility of the constraints in the metaobjects is checked w.r.t. the TBox. To carry out the satisfiability test, our method relies on the calculus explained in [4]. The computational properties of this calculus for the $\mathcal{AL}$-languages family are also discussed in [4].
In the second phase, the metaobjects associated to external facts are inserted in the subcontexts and the deductive tree is contracted by means of the context operations (these operations are explained in [3]). Thus, all the metaobjects generated from external facts are collected in the context associated with the IC.

References

1. Levy, A.Y., Rousset, M.: Verification of knowledge bases on containment checking. Artificial Intelligence **101** (1998) 227–250
2. Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A.: AL-log: Integrating datalog and description logics. Journal of Intelligent Information Systems **10** (1998) 227–252
3. 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)
4. Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W.: The complexity of concept languages. Technical Report RR-95-07, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH Erwin-Schrödinger Strasse Postfach 2080 67608 Kaiserslautern Germany (1995)
5. Horrocks, I.: Using an expressive description logic: FaCT or fiction? In Cohn, A.G., Schubert, L., Shapiro, S.C., eds.: Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98), Morgan Kaufmann Publishers, San Francisco, California (1998) 636–647