A Logic Based Approach to the Static Analysis of Production Systems

Jos de Bruijn and Martín Rezk
{debruijn,rezk}@inf.unibz.it
Free University of Bozen-Bolzano

I. Introduction

Production systems (PS) are one of the oldest knowledge representation paradigms in artificial intelligence, and are still widely used today, [1] for example to preserve consistency in databases, AI planning, etc. Such a system consists of a set of rules r of the form "if *condition*$_r$ then *action*$_r$", a working memory, which contains the current state of knowledge, and a rule interpreter, which executes the rules and makes changes in the working memory, based on the actions in the rules.

In general rule-based systems are administered and executed in a distributed environment where the rules are interchanged using standardized rule languages, e.g. RIF, RuleML, SWRL. The new system obtained from adding (or removing) the interchanged rules need to be consistent, and some properties be preserved, e.g. termination. In this work we address the static analysis of such production systems, which means deciding properties like termination and confluence (all runs of the systems terminate with the same working memory) . We propose using logics and their reasoning techniques from the area of software specification and verification, in particular μ-calculus [1] and fixed-point logic (FPL) [2].

Given a working memory (a set of facts, i.e., ground atomic formulas), the rule interpreter applies rules in three steps: (1) *pattern matching*, (2) *conflict resolution*, and (3) *rule execution*. In the first step, the interpreter decides – nowadays typically using the RETE algorithm [3] – for each rule r_i and for each variable substitution σ_j whether r_i can be applied in the working memory using σ_j, i.e., whether the working memory satisfies $\sigma_j(\textit{condition}_{r_i})$. This step returns all pairs (r_i, σ_j) such that r_i can be applied using σ_j; this set is called the conflict resolution set. In step (2), the interpreter chooses zero or one pair from the conflict resolution set; in case the set is empty or no pair is chosen, the system terminates. In the last step, the working memory is updated following the additions and removals in the action part of the selected rule. The interpreter then starts again with step (1).

The operational semantics of production systems makes it difficult to analyze their behavior. Therefore, it is desirable to use a formalism with a declarative semantics for static analysis. We use two well-known logics that are frequently used in the area of software verification. For the analysis of propositional systems we use μ-calculus, which is a modal logic extended with the least and greatest fixpoint operators, and for which common reasoning tasks, such as entailment, are decidable in exponential time. For the first-order systems we use fixed-point logic (FPL), an extension of FOL with least and greatest fixpoint operators. Even though reasoning with FPL is not decidable in the general case, there are decidable subsets [4].

Our main contributions with this paper are as follows. We present an embedding of propositional production systems into μ-calculus and show how this embedding can be used for the static analysis of production systems. We then present an embedding of first-order production systems in fixed-point logic, show how the embedding can be used for reasoning over the production system, and discuss two decidable cases.

We use properties of these logics to derive (un)decidability and complexity results for deciding properties such as termination and confluence of production systems. The embedding of first-order production systems into FPL serves as a starting point for investigating further decidable subsets (e.g., based on the guarded fragment [4]), in particular when considering further strategies that limit the choice in the conflict resolution step (2) of the rule application – for example, such strategies may guarantee termination, and thus finite models of the embedding.

II. Propositional Production Systems

In this section we present formal definitions of propositional production systems, and then the axiomatization needed to check properties on PS.

Definition 1: A *Generic Production System* (GPS) is a tuple $PS = (Prop, L, R)$, where $Prop$ is a finite set of propositions, representing the set of potential facts, L is a set of rule labels, and R is a set of *rules*, which are statements of the form

$$r: \text{ if } \phi_r \text{ then } \psi_r$$

where $r \in L$, ϕ_r is a propositional formula, and $\psi_r = a_1 \wedge \cdots \wedge a_k \wedge \neg b_1 \wedge \cdots \wedge \neg b_l$, with $a_i \neq b_j$ ($a_i, b_j \in Prop$) signifying the propositions added, respectively removed by the rule

In the following, let $PS = (Prop, L, R)$ be a production system. A *Working Memory* $WM \subseteq Prop$ for PS is a set of propositions.

A rule r is *fireable* in a working memory WM if $WM \models \phi_r$ and $WM' = WM \cup \psi_r^{add} \setminus \psi_r^{remove} \neq WM$.

A *concrete production system* (CPS) is a pair (PS, WM_0), where WM_0 is a working memory.

Definition 2: A *computation tree* $CT^{PS}_{WM_0}$ for a CPS (PS, WM_0) is a $(Prop \cup L)$-labeled tree (T, V) such that the root of T is 0, $V(0) = WM_0$, and each branch represents a run of the system.

A. Axiomatization

The existence of a formal description of any language is a prerequisite to any rigorous method of proof, validation, or ver-

[1] `http://www.jessrules.com/` `http://clipsrules.sourceforge.net/` `http://www.ilog.com/products/jrules/`

ification. Here we present (due to lack of space) the general idea of the axiomatization of production systems in μ-calculus. In the following subsections we will show how this axiomatization can be used for reasoning about production systems.

We first define the necessary components of the formula comprising the axiomatization. These components (nine axioms) encode the constrains and requirements in the relation between one state and its successors depending if it is an intermediate state in the execution of the PS, or a state representing the end of a run. We also provide an axiom to solve the frame problem for this case. A greatest fix point composed of these components restricts the models to the ones which are bisimilar to a computation tree (CT). The states in the models of the axiomatization can be seen as nodes in the computation tree. Abstractly the μ-calculus formula that captures the production system PS looks like:

$$\Phi_{PS} = [(RootAlone) \vee (ExistsApplicableRules \wedge \Box(\nu.X.(\textbf{intermediate} \vee \textbf{end}) \wedge \Box X)))]$$

Where the greatest fix point requires every successor of the root to be an intermediate node in the run (satisfying a set the axioms, for example, a rule was applied, it has a predecessor in which a precondition of the rule holds) or the last node of a terminating run (satisfying other set of axioms encoding that it has no successor, no rule can be applied, etc).

We now proceed to prove bisimilarity between the models of Φ_{PS} and the computation trees of PS. We will exploit this result later for reasoning about PS.

Theorem 1: Given a Production system $PS = (Prop, L, R)$, a starting working memory WM_0, and the formula Φ_{PS}, a Kripke structure $K = (S, R, V)$ is a model of Φ_{PS} iff there is a working memory WM for PS such that there is an $s \in S$ and (K, s) is bisimilar to $(CT^{PS}_{WM}, 0)$

An analogous theorem state the bisimilarity relation between CPS and an specific $CT^{PS}_{WM_0}$.

B. Deciding Properties of Production Systems

Typical properties of production systems one would like to check are termination and confluence of the system. However, one could imagine many additional properties of interest, e.g., redundancy of rules (useful in the design of the system). Here, due to lack of space, we will discuss just about termination and confluence. Termination can be encoded in μ-Calculus as:

$$(\mu.X.\Box X)$$

and Confluence

$$\bigwedge_{q_i \in Prop} (\mu.X.(\Box \bot \wedge q_i) \vee \Diamond X) \rightarrow (\nu.X.(\Box \bot \rightarrow q_i) \wedge \Box X)$$

Properties termination and confluence hold for a generic/concrete production system PS iff Φ_{PS} entails the μ-calculus formula encoding them. From the fact that Φ_{PS} is polynomial in the size of PS and the fact that μ-calculus entailment can be decided in exponential time, we conclude that these properties can be decided in exponential time, both on generic and concrete production systems.

III. First Order Production Systems

We now consider the case of production systems with variables. In the first order case Interpretations in FPL are first-order structures; therefore, we capture the structure of the computation tree using the binary predicate R, and we divide the domain into two parts: the nodes of the tree, i.e., the states (A), and the objects in the working memories (U). The arity of the predicates in $P \cup L$ is increased by one, and the first argument of each predicates will signify the state; $p(y, x_1, \ldots, x_n)$ intuitively means that $p(x_1, \ldots, x_n)$ holds in state y.

Analogous to the propositional case, we defined a formula that captures the behavior of PS. The most notable differences with the propositional axiomatization is that in the first order case we have a set of "foundational" axioms to describe the domain, and that in the propositional case, we could require that a fireable rule is applied at least once, but it could be applied several times. In the first-order case, we can require a fireable rule to be applied exactly once. We can therefore obtain a stronger correspondence (compared with the one in Theorem 1) between computation trees and Kripke models: they are essentially isomorphic.

FPL is undecidable, but under certain constrains we can still reduce the problem to a decidable logic.

Proposition 2: Let PS be an FO production system such that rules are quantifier-free and the set of constants are finite, and let WM be a working memory. Then, the properties termination and confluence can be decided in double exponential time, on both PS and (PS, WM).

IV. Conclusions and Future Work

In this paper we presented an embedding of propositional production systems into μ-calculus, and first-order production systems into fixed-point logic. We exploited the fixpoint operator in both logics to encode properties of the system over time. One of the advantages of our encodings is the strong correspondence between the structure of the models and the runs of the production systems, which enables straightforward modeling of properties of the system in the logic. We plan to extend the work presented in this paper in a number of directions. We plan to extend both the propositional and first-order case with additional conflict resolution strategies, e.g., based on rule priorities. We plan to extend the first-order case with object invention, i.e., the rules may assert information about new (anonymous) objects; this is strongly related to existential quantification in logic. Another topic we plan to address are new decidable fragments of our first-order encoding, in particular restricting the conditions and possibly the working memory, and conflict resolution strategies in order to exploit the guarded fragment of FPL [4], as well as translations to monadic second-order logic over trees; both fragments are known to be decidable.

References

[1] Kozen, D.: Results on the propositional μ-calculus. In: Proceedings of the 9th Colloquium on Automata, Languages and Programming, London, UK, Springer-Verlag (1982) 348–359

[2] Gurevich, Y., Shelah, S.: Fixed-point extensions of first-order logic. Symposium on Foundations of Computer Science **0** (1985) 346–353

[3] Forgy, C.: Rete: A fast algorithm for the many patterns/many objects match problem. Artif. Intell. **19**(1) (1982) 17–37

[4] Grädel, E.: Guarded fixed point logics and the monadic theory of countable trees. Theor. Comput. Sci. **288**(1) (2002) 129–152