A Logic Based Approach to the Static Analysis of Production Systems

Jos de Bruijn and Martín Rezk

KRDB Research Center
Free University of Bozen-Bolzano, Italy
{debruijn,rezk}@inf.unibz.it

Abstract. In this paper we present an embedding of propositional production systems into μ-calculus, and first-order production systems into fixed-point logic, with the aim of using these logics for the static analysis of production systems with varying working memories. We encode properties such as termination and confluence in these logics, and briefly discuss which ones cannot be expressed, depending on the expressivity of the logic. We show how the embeddings can be used for reasoning over the production system, and use known results to obtain upper bounds for special cases. The strong correspondence between the structure of the models of the encodings and the runs of the production systems enables the straightforward modeling of properties of the system in the logic.

1 Introduction

Production systems (PS) are one of the oldest knowledge representation paradigms in artificial intelligence, and are still widely used today[1]. 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. 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].

In this work we consider rules in which conditions are first-order logic (FOL) formulas with free variables and the actions are *additions* and *removals* of atomic formulas. We also consider the special case of variable-free, i.e., propositional rules. We note here that in case a limited number of constant symbols is available and the rules are quantifier-free, the first-order case can be reduced to the propositional case through grounding, i.e., replacing each rule with all possible ground variable substitutions.

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

The working memory of a production system is a set of facts, i.e., ground atomic formulas. Given a working memory, 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 $\sigma_j(\mathit{condition}_{r_i})$. 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 non deterministically chooses a pair from the conflict resolution set; in case the set is empty 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).

We note here that the choice of conflict resolution strategy affects certain properties of the production system – for example, if one chooses a conflict resolution strategy that allows each rule to be executed at most once, the system is guaranteed to terminate. Therefore, the conflict resolution strategy of a production system must be known when performing static analysis. In the present paper we assume a simple conflict resolution strategy: the rule interpreter arbitrarily selects one pair (r_i, σ_j) from the resolution sets such that applying $\sigma_j(action_{r_i})$ to the working memory yields an updated working memory that is different from the current one. We leave consideration of further conflict resolution strategies for future work.

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. In addition, we are interested in deciding properties of production systems for which the initial working memory varies – e.g., we want to be able to decide whether the execution of a set of rules terminates: no matter what the start state is. 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.

The papers is further structured as follows. We review related work in Section 2 and give preliminary definitions in Section 3. We present our embedding of propositional production systems in μ-calculus and show how it can be used for static analysis in Section 4. In Section 5, we present our embedding of first-order systems in FPL. We conclude and discuss future work in Section 6.

2 Related Work

We consider two streams of related work: *action languages and planning* and *rules in active databases*.

The situation calculus [5] is one of the classic formalisms for representing action and change in artificial intelligence. One might thus consider it an alternative formalism that may also be used for capturing production systems. A distinguishing feature between situation calculus, on the one hand, and μ-calculus and our use of FPL, on the other, is that in the former there is a notion of situation – essentially a term capturing the history of the actions – and in the latter there is a notion of state – essentially a collection of all the facts that hold at a given point in time. Arguably, the latter are conceptually a better match with the notion of working memory in production systems. Nonetheless, the situation calculus has been used in the context of production systems by [6]. Specifically, [6] used logic programs with the stable model semantics and situation calculus notation for characterizing production systems, and deciding properties such as confluence and termination. A notable difference between our approach and the one base on stable model semantics, is that, in general, we allow the initial working memory to vary – we can do that since μ-calculus and FPL allow reasoning over all possible models.

In the planning domain and STRIP-like languages, one assumes a set of operators with preconditions and actions, an initial database and a goal. The planning problem is to find a sequence of operations that updates the initial database so that the goal holds. One might reuse planning results for the static analysis of production systems by considering properties of systems as goals and finding a plan involving rule applications as atomic actions (for example $G = p$ meaning that p eventually holds). This problem has been approached from the logic point of view in many ways, many of them (that we are aware of) started with [7] approaching planning as a satisfiability problem. Reiter [8] allows first order sentences for the goal, which makes the problem undecidable. In [9,10], the authors address the propositional case with LTL. The main problem in trying to apply these works to PS, is that in the planning problem, they need to find *one* sequence which satisfies the goal, or the *absence* of a sequence. This lead the research to linear temporal logic, where basic properties like confluence can be expressed in a first order extension (but maybe encoded in the propositional case) and some other properties, like **[PE5]** in Section 4.2 can not be neither expressed nor encoded. On the other hand, if we consider the operators as the PS's rules, the present work can be used to solve to planning problem using **[PE4]** (in Section 4.2) and replacing p by the Goal.

In [11], the authors present a new logic ($\mathcal{DIFR}$) which is an extension of PDL that can encode propositional situation calculus. They present a formal framework for modeling, and reasoning about actions. Consequently, each particular problem has to be modeled ad-hoc. In the present work we model not just the conditions and effect of an action, but several specific features of Production systems like strategies, constrains, and the behavior of the system in time. We provide an axiomatization of PS (Section 4), and a formal proof of the correspondence with the set of runs of a PS, and the models of our axiomatization (Theorem 1). This link is required to do formal verification of properties of PS, using the models of the axiomatization. The choice of μ-calculus over $\mathcal{DIFR}$ for modeling has been based on two points: First, certain properties of interest, like finiteness of runs (among others), cannot be expressed in $\mathcal{DIFR}$, while they can be expressed in μ-calculus (see Section 4.2). Second, in Section 5 we extend the propositional case, and we model PS with variables, First Order Production Systems (*FO-PS*). This model is in Fix Point Logic, which can be seen as a first order extension of μ-calculus, therefore the choice of μ-calculus makes the path from the propositional PS to FO-PS more understandable.

Rules in active databases are strongly related to production rules. While production rules are condition-action rules, active databases contain event-condition-action rules; there are external events that may trigger firing of rules. The techniques we are aware of that have been proposed for the static analysis of such rules ([12,13,14]) are based on checking properties of graphs, where nodes are rules, and an edge between r_1 and r_2 means that the action of r_1 can trigger the firing of r_2. The general problem, where conditions are arbitrary SQL queries, is (unsurprisingly) known to be undecidable [15]; an analogous result for our case is stated in Theorem 4. In [12], [13], and [14] the authors study sufficient conditions for deciding termination and confluence. In contrast, our embeddings in μ-calculus and FPL are used to find sufficient and *necessary* conditions for deciding these and other properties for classes of production systems.

3 Preliminaries

***μ*-Calculus.** Let Var be a (infinite) set of variable names, typically written $Y, Z \ldots$ and let $Prop$ be a set of atomic propositions, typically written p, q. μ-calculus extends propositional logic with the modal operator $\Diamond$ and with formulas of the form $\mu.Z.\phi(Z)$, where $\phi(Z)$ is a μ-calculus formula in which the variable Z occurs positively, i.e., under an even number of negations.

As usual, $\Box\phi$ is short for $\neg\Diamond\neg\phi$ and $\nu.Z.\phi(Z)$ is short for $\neg\mu.Z.\neg\phi(\neg Z)$.

A *Kripke structure* is a tuple $K = (S, R, V)$, where S is a non-empty (possibly infinite) set of states, $R \subseteq S \times S$ a binary relation over S, and $V : S \rightarrow 2^{Prop}$ assigns to each proposition $p \in Prop$ a (possibly empty) set of states.

A valuation $\mathcal{V} : S \rightarrow 2^{V}ar$ assigns to each variable a set of states. For a valuation $\mathcal{V}$, a variable Y and a set S, we denote by $\mathcal{V}[Y \leftarrow S]$ the valuation obtained from $\mathcal{V}$ by assigning S to Y.

Given a Kripke structure $K = (S, R, V)$, we define the set of states satisfying a formula ϕ, relative to a valuation $\mathcal{V}$, denoted $\phi^K(\mathcal{V})$, as follows:
– $p^K(\mathcal{V}) = \{s \in S \mid p \in V(s)\}$ for propositions p, $Y^K(\mathcal{V}) = \mathcal{V}(Y)$ for variables Y,
– $(\phi_1 \wedge \phi_2)^K(\mathcal{V}) = (\phi_1)^K(\mathcal{V}) \cap (\phi_2)^K(\mathcal{V})$,
–$(\neg\phi)^K(\mathcal{V}) = S \backslash (\phi)^K(\mathcal{V})$,
– $(\Diamond\phi)^K(\mathcal{V}) = \{s \in S \mid \exists s' \in \phi^K(\mathcal{V}).(s, s') \in R$, and
– $(\mu.Z.\phi(Z))^K(\mathcal{V}) = \bigcap\{S' \subseteq S \mid \phi^K(\mathcal{V}[Z \leftarrow S']) \subseteq S'\}$.

A μ-calculus formula ϕ is *satisfiable* iff there exists a structure K s.t. $(\phi)^K(\mathcal{V}) \neq \emptyset$, for every valuation $\mathcal{V}$; in this case K is a *model* of ϕ. A formula ϕ *entails* a formula ψ iff $(\phi \wedge \neg\psi)$ is not satisfiable.

Fixed Point Logic. Fixed Point Logics FPL [4]) extend standard first order logic (with equality and without function symbols) with least fixed-point formulas of the form $[\mu W.\boldsymbol{x}.\psi(W, \boldsymbol{x})](\boldsymbol{x})$, where W is a k-ary relation symbol (a second order variable), $\psi(W, \boldsymbol{x})$ contains only positive occurrence of W and its free first-order variables are in $\boldsymbol{x}$. As usual, the greatest fixed-point formula $[\nu W.\boldsymbol{x}.\psi(W, \boldsymbol{x})](\boldsymbol{x})$ is short for $\neg[\mu W.\boldsymbol{x}.\neg\psi(\neg W, \boldsymbol{x})](\boldsymbol{x})$.

In order to obtain the necessary correspondence with the constants employed in the production system, we assume *standard names*. Let C be the set of constants. Then, all interpretations are of the form $\mathcal{M} = \langle C, \cdot^{\mathcal{M}} \rangle$ and we have that $c^{\mathcal{M}} = c$, for every $c \in C$.

Given a structure $\mathcal{M} = \langle \Delta, \cdot^{\mathcal{M}} \rangle$ providing interpretations for all the free second order variables in ψ, except W, the formula $\psi(W, \boldsymbol{x})$ defines an operator on k-ary relations $W \subseteq A^K$:

$$\psi^{\mathcal{M}} : W \mapsto \psi^{\mathcal{M}}(W) := \{\boldsymbol{a} \in \Delta^k : \mathcal{M} \models \psi(W, \boldsymbol{a})\}$$

Since W occurs only positively in ψ, this operator is monotone and therefore has a least fixed point $LFP(\psi^{\mathcal{M}})$. We then define

$$\mathcal{M}, B \models [\mu W.\boldsymbol{x}.\psi(W, \boldsymbol{x})](\boldsymbol{x}) \text{ iff } B(\boldsymbol{x}) \in LFP(\psi^{\mathcal{M}})$$

for interpretation $\mathcal{M}$ and first-order variable assignment B.

Satisfaction and entailment are then defined in the usual way.

4 Propositional Production Systems

An intensively investigated area of temporal logic is automatic verification of propositional temporal logic properties of finite state systems. As is well known, semantic properties of programs written in Turing-complete languages are undecidable, thus a full verification of such a program intrinsically includes hand work. However, there are many applications from components of microprocessors to communication protocols, which work with finite data domains, and which can be represented by finite state transition systems. Even if a system is by nature infinite, sometimes the possibility exists to abstract from infinitary aspects and obtain a finitary representation of the relevant aspects of the behavior. Propositional logic succeeds to specify the static properties of finite state systems, i.e. the properties of the states. Moreover, if we have finite state systems on the one hand, and propositional temporal logics, i.e. propositional logics amalgamated with temporal constructs, on the other hand, then automatic verification is possible. This means, there exist programs taking a description of a finite state system and a propositional temporal specification as input and return as result, whether the system satisfies the temporal property.

We first present formal definitions of propositional production systems.

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*

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

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, such that every rule has a distinct label and $L \cap Prop = \emptyset$. We define $\phi_r^{add} = \{a_1, \ldots, a_k\}$ and $\phi_r^{remove} = \{b_1, \ldots, b_l\}$.

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. As an abuse of notation, we use WM for both the working memory and the propositional valuation induced by it (i.e., $WM \models p$ iff $p \in WM$).

A rule r is *fireable* in a working memory WM if $WM \models \phi_r$ and $WM' = WM \cup \psi_r^{add} \setminus \psi_r^{remove} \neq WM$.[2]

A *concrete production system* (CPS) is a pair (PS, WM_0), where WM_0 is a working memory. For a set of symbols Σ, a *Σ-labeled tree* is a pair (T, V), where $T \subseteq \mathbb{N}^+$ is such that if $x.c \in T$, then also $x \in T$ and $V : T \rightarrow 2^{\Sigma}$ maps each node to a set of symbols in Σ.

Definition 2. *A* computation tree $CT_{WM_0}^{PS}$ *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 for each node $n \in T$ and every rule r that is fireable in the working memory $WM = V(n) \cap Prop$, there is a child node $n' \in T$ of n such that $V(n') = WM' \cup \{r\}$, with $WM' = WM \cup \psi_r^{add} \setminus \psi_r^{remove}$. There are no other nodes in $CT_{WM_0}^{PS}$.*

Note that the rule label r in the label of each node represents the rule that has been fired to obtain the current node from the parent. Note also that $CT_{WM_0}^{PS}$ is unique up to isomorphisms, and so we may speak about *the* computation tree of a CPS. A *run* of a CPS is a branch in $CT_{WM_0}^{PS}$. A run is *terminating* if it is finite.

In the remainder of this section we present an axiomatization of production systems in μ-calculus and show how this can be used for static analysis.

4.1 Axiomatization

The existence of a formal description of any language is a prerequisite to any rigorous method of proof, validation, or verification. Here we present an 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.

In the following, let $PS = (Prop, L, R)$ be a generic production system and let WM_0 be a working memory. We first define the necessary components of the formula

[2] Note that we assume here a simple conflict resolution strategy, namely a rule is only fired if it brings about a change in the working memory.

comprising the axiomatization. These components 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. 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. We assume that b does not appear in $Prop \cup L$.

Root. The current state represents the root of the CT.

$$b \wedge \bigwedge_{r \in L} \neg r$$

RApp. Rule application.

$$(\bigwedge_{r \in L}(r \rightarrow \psi_r))$$

Appl. If a rule is applied, it must be applicable.

$$(\bigwedge_{r \in L} \Diamond r \rightarrow \phi_r \wedge \neg \psi_r)$$

Frame. Frame axiom: if q holds, it holds in the next state unless q is removed and if $\neg q$ holds, $\neg q$ holds, unless q is added.

$$(\bigwedge_{q \in Prop}(q \rightarrow \Box(q \vee (\bigvee_{r \in L . q \in \psi_r^{remove}} r))) \wedge \\ (\neg q \rightarrow \Box(\neg q \vee (\bigvee_{r \in L . q \in \psi_r^{add}} r))))$$

NoFireable. No rule is fireable and there is no successor.

$$[(\bigwedge_{r \in L}(\phi_r \rightarrow \psi_r)) \wedge (\Box \perp]$$

Fireable. At least one rule is fireable and there is a successor.

$$(\bigvee_{r \in L} \phi_r \wedge \neg(\psi_r)) \wedge \Diamond \top$$

Complete. If a rule is fireable, it is applied in some successor states.

$$(\bigwedge_{r \in L}(\phi_i \wedge \neg(\psi_r)) \rightarrow \Diamond r)$$

1Rule. Exactly one rule is applied.

$$(\bigvee_{r \in L} r \wedge (\bigwedge_{r \in L}(r \rightarrow \neg \bigvee_{r' \in L \& r' \neq r} r')))$$

WM. (Optional) The initial working memory holds.

$$\bigwedge_{q \in WM_0} q \wedge \bigwedge_{q \in Prop \setminus WM_0} \neg q$$

The axiom **Root.** captures the root of the computation tree. We now define the axioms which capture intermediate and end (i.e., leaf) nodes.

Intermediate $=$ **RApp.** $\wedge$ **1Rule.** $\wedge$ **Appl.** $\wedge$ **Frame.** $\wedge$ **Fireable.** $\wedge$ **Complete.** $\wedge \neg b$
End $=$ **RApp.** $\wedge$ **1Rule.** $\wedge$ **Frame.** $\wedge$ **NoFireable.** $\wedge \neg b$

We now define the μ-calculus formula that captures the production system PS:

$$\Phi_{PS} = [(\textbf{Root.} \wedge \textbf{NoFireable.}) \vee (\textbf{Root.} \wedge \textbf{Appl.} \wedge \textbf{Frame.} \wedge \\ \textbf{Complete.} \wedge \textbf{Fireable.} \wedge \Box(\nu.X.(\textbf{Intermediate} \vee \textbf{End}) \wedge \Box X)))]$$

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.

A *bisimulation* between two pointed Kripke structures, $K = ((S, R, V), s_0)$ and $K' = ((S', R', V'), t'_0)$ is a relation $Z \subseteq S \times S'$ such that:
– $(s_0, t'_0) \in Z$
– if $(s_i, t'_i) \in Z$, then $p \in V(s_i)$ iff $p \in V'(t'_i)$, for every proposition p,
– if $(s_i, t'_i) \in Z$ and $(s_i, s') \in R$ implies that there is a $t' \in S'$ such that $(t'_i, t') \in R'$ and $(s', t') \in Z$
– if $(s_i, t'_i) \in Z$ and $(t_i, t') \in R'$ implies that there is a $s' \in S$ such that $(s_i, s') \in R$ and $(s', t') \in Z$

We view a computation tree (T, V), with 0 being the root, also as a Kripke structure $K = (T, R, V')$, where $V'(0) = V(0) \cup \{b\}$, $V'(n) = V(n)$ for $n \neq 0$, and $(n, n') \in R$ iff $n.n' \in T$.

Theorem 1. *Given a Production system $PS = (Prop, L, R)$, a starting working memory WM_0, and the formula Φ_{PS}.*

1. *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 $(CT^{PS}_{WM}, 0)$*
2. *A Kripke structure $K = (S, R, V)$ is a model of $\Phi_{PS} \wedge$ **WM.** iff there is an $s \in S$ such that (K, s) is bisimilar to $(CT^{PS}_{WM_0}, 0)$.*

Proof (Sketch). We start with 2. If $K = (S, R, V)$ is a model of $\Phi_{PS} \wedge$ **WM.** then there is at least a node s_0 s.t. $K, s_0 \models$ **Root.** $\wedge$ **WM.**. We start defining the bisimulation Z: $(s_0, 0) \in Z$. Now we have to define bisimulation in such a way that $(s_i, xi) \in Z$ iff $(s_j, x) \in Z$ and s_i successor of s_j, $x.i$ is the successor of x, and $s_i, x.i$ agree on the proposition constants. Let's take a node s_i successor of s_j such that $(s_j, x) \in Z$ and $V(s_j) = V(x)$. By **Complete.** we know that there is at least one successor for each applicable rule in s_j, and by **RApp.** and **Frame.**, we know that the label of the successor is just the result of the rule application. By **Fireable.** we know that the precondition of every successor hold, therefore, for each successor of s_j in K, we have a successor of x in CT_{ps} (note it can be many-to-one) with the same label, so $(s_i, xj) \in Z$ for every s_i and some xj which is determined by the rule label. The other direction is analogous.

Now, the proof of 1 is straightforward: if we have a model K of Φ_{PS} where some state s_0 (as defined above) is the set of proposition WM, we know that PS with WM as the initial working memory is bisimilar to K by point 2. The converse is analogous.

4.2 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 additional properties of interest, e.g., redundancy of rules (useful in the design of the system). In this section we showcase a number of properties that we feel might be of interest, and that can be reduced to μ-calculus satisfiability or entailment checking, using the axiomatization of the previous section.

The properties stated below are defined for both generic and concrete production systems. **PE**i are the properties that can be decided by checking entailment . With $\phi_{\mathbf{PE}i}$ we denote the formula associated with **PE**i.

PE1. All runs are finite (i.e., Termination)

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

PE2. All runs terminate with the same working memory (Confluence)

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

PE3. There is a fireable rule in the initial working memory

$$(\bigvee_{r \in R} \phi_r)$$

PE4. A proposition p eventually holds in some run.

$$(\mu.X.(p \wedge (\nu.Y.p \wedge \Diamond Y)) \vee \Diamond X)$$

PE5. A proposition p eventually holds for ever in every run.

$$(\mu.X.(p \wedge (\nu.Y.p \wedge \Box Y)) \vee \Box X)$$

PE6. Some rule r is never applied

$$\neg(\mu.X.\Diamond X \vee r)$$

PE7. All rules are applied in every run

$$\bigwedge_{r_i \in R}(\mu.Z.r_i \vee \Box Z \wedge \Diamond \top)$$

We now show how deciding the above properties can be reduced to μ-calculus entailment checking, by exploiting Theorem 1.

Theorem 2. *A property* ***PE****i, for* $i \in \{\mathbf{1}, \ldots, \mathbf{7}\}$ *holds for a generic production system* PS *iff* Φ_{PS} *entails* ***PE****i and* ***PE****i holds for a concrete production system* (PS, WM_0) *iff* ***WM.*** $\wedge\, \Phi_{PS}$ *entails* $\phi_{\mathbf{PE}i}$.

Note that when considering concrete production systems, some of the mentioned properties (e.g., **PE3**) can be decided by simply running the system. However, certain other properties (e.g., termination) cannot.

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 immediately obtained the following complexity results.

Proposition 1. *The properties* ***PE1-7*** *can be decided in exponential time, both on generic and concrete production systems.*

5 First Order Production Systems

We now consider the case of production systems with variables.

Definition 3. *A* Generic FO-Production System *is a tuple $PS = (\tau, L, R)$, where*
– $\tau = (P, C)$ is a first-order signature, with P a set of predicate symbols, each with an associated nonnegative arity, and C a nonempty (possibly infinite) set of constant symbols,
– L is a set of rule labels, and
– R is a set of rules, which are statements of the form

$$r: \textit{ if } \phi_r(\boldsymbol{x}) \textit{ then } \psi_r(\boldsymbol{x})$$

where $r \in L$, ϕ_r is an FO formula with free variables $\boldsymbol{x}$ and $\psi_r(\boldsymbol{x}) = (a_1 \wedge \cdots \wedge a_k \wedge \neg b_1 \wedge \cdots \wedge \neg b_l)$, where $a_1, \ldots, a_k, b_1, \ldots, b_l$ are atomic formulas with free variables among $\boldsymbol{x}$, such that no a_i and b_j share the same predicate symbol, each rule has a distinct label and $L \cap Prop = \emptyset$. We define $\phi_r^{add} = \{a_1, \ldots, a_k\}$ and $\phi_r^{remove} = \{b_1, \ldots, b_l\}$.

In the following, let $PS = (\tau, L, R)$ be an FO-production system. With AT we denote the set of equality-free ground atomic formulas (atoms) of τ. A working memory WM for PS is a subset of AT. As an abuse of notation, we use WM to denote both the working memory and the first-order structure induced by the working memory, i.e., the domain of WM is C, $c^{WM} = c$, for any $c \in C$, and $\boldsymbol{c} \in p^{WM}$ iff $p(\boldsymbol{c}) \in WM$ for any $p(\boldsymbol{c}) \in AT$.

A variable substitution $\mathcal{S}$ is a mapping from variables to constants in C. The application of a variable substitution to a term or formula φ, written $\mathcal{S}(\varphi)$, is defined in the usual way. A rule is *fireable* in a working memory WM using a substitution $\mathcal{S}$ if $WM \models \mathcal{S}(\phi_r)$ and $WM' = WM \cup \mathcal{S}(\psi_r^{add}) \setminus \mathcal{S}(\psi_r^{remove}) \neq WM$.

A *concrete FO-production system* is a pair (PS, WM_0), where WM_0 is a working memory. We view rule labels $r \in L$ also as n-ary predicates, where n is the number of free variables in the condition ϕ_r; with AL we denote the set of ground atoms constructed from the predicate symbols in L and the constants in C.

Definition 4. *A* computation tree $CT_{WM_0}^{PS}$ *for a (PS, WM_0) is an $(AT \cup AL)$-labeled tree (T, V) such that the root of T is 0, $V(0) = WM_0$, and for each node $n \in T$, every rule r, and every variable substitution $\mathcal{S}$ such that r is fireable in the working memory $WM = V(n) \cap Prop$ using $\mathcal{S}$, there is a child node $n' \in T$ of n such that $V(n') = WM' \cup \{\mathcal{S}(r(\boldsymbol{x}))\}$, with $WM' = WM \cup \mathcal{S}(\psi_r^{add}) \setminus \mathcal{S}(\psi_r^{remove})$. There are no other nodes in $CT_{WM_0}^{PS}$.*

A *run* of (PS, WM_0) is a branch of $CT_{WM_0}^{PS}$. A run is *terminating* if it is finite.

In the remainder of this section we discuss special cases of FO production systems that can be reduced to propositional production systems, we present an axiomatization of general FO production systems in fixed-point logic (FPL), and show how static analysis can we reduced to reasoning with FPL. This axiomatization will be a starting point for future investigation of decidable fragments.

5.1 Grounding FO Production Systems

The *grounding* of an FO production system $PS = (\tau, L, R)$, denoted $gr(PS)$, is obtained from PS by replacing each rule r : if $\phi_r(\boldsymbol{x})$ then $\psi_r(\boldsymbol{x})$ with a set of rules $\mathcal{S}(r(\boldsymbol{x}))$: if $\mathcal{S}(\phi_r(\boldsymbol{x}))$ then $\mathcal{S}(\psi_r(\boldsymbol{x}))$, for every substitution $\mathcal{S}$ of variables with constants in C.

Clearly, for any working memory WM, the computation trees of (PS, WM) and $(gr(PS), WM)$ are the same. Also, if the rules in PS are quantifier-free, $gr(PS)$ can be seen as a propositional generic production system – in the absence of variables, atomic formulas are essentially propositions. This allows us to apply some of the results for the propositional case to FO production systems.

We first exploit the fact that if the set of constants C is finite, the grounding $gr(PS)$ is finite, and its size exponential in the size of PS.

Proposition 2. *Let $PS = (\tau, L, R)$ be an FO production system such that R is quantifier-free and C is finite, and let WM be a working memory.*[3] *Then, the properties **PE1-7** can be decided in double exponential time, on both PS and (PS, WM).*

When considering concrete FO production systems, i.e., the initial working memory is given, we can also exploit grounding, provided the conditions in the rules are *domain-independent* (cf. [16]): a first-order formula with n free variables $\phi(\boldsymbol{x})$ is domain-independent iff whenever $\mathcal{M} = \langle \Delta, \cdot^{\mathcal{M}} \rangle$ and $\mathcal{M}' = \langle \Delta, \cdot^{\mathcal{M}'} \rangle$ are structures, $\mathcal{M}$ is a substructure of $\mathcal{M}'$, and the interpretations functions are identical ($\cdot^{\mathcal{M}'} = \cdot^{\mathcal{M}}$), then for all $a_1 \in \mathcal{M}', \dots, a_n \in \mathcal{M}'$:

$$\mathcal{M}' \models \phi(a_1 \dots a_n) \leftrightarrow a_1 \in \mathcal{M} \wedge \dots \wedge a_n \in \mathcal{M} \wedge \mathcal{M} \models \phi(a_1 \dots a_n)$$

If all conditions are domain-independent and the initial working memory WM_0 is given, one only needs to consider grounding with the constants appearing in (PS, WM_0). Examples of domain-independent formulas are conjunctions of literals such that each variable occurs in a positive literal.

Proposition 3. *Let $PS = (\tau, L, R)$ be an FO production system such that R is quantifier-free and for every rule $r \in R$ holds that $\phi_r(\boldsymbol{x})$ is domain-independent, and let WM be a working memory. Then, the properties **PE1-7** can be decided in double exponential time, on (PS, WM).*

5.2 Axiomatizing FO Production Systems

In the remainder we assume that the signature of each of the production systems contains a countably infinite set of constant symbols. In our μ-calculus axiomatization for the propositional case the structure of the computation tree was reflected in the accessibility relation of the Kripke models. Interpretations in FPL are first-order structures; therefore, we capture the structure of the computation tree using the binary predicate R,

[3] Note that if C is finite, the existential quantifier could be replaced with a disjunction of all possible ground variable substitutions; analogous for universal quantifier. In this case, the grounding would be double exponential.

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; $p(y, x_1, \dots, x_n)$ intuitively means that $p(x_1, \dots, x_n)$ holds in state y.

In the remainder, let $PS = (\tau, L, R)$ be a generic FO production system and let WM_0 be a working memory. We first define the *foundational axioms*, which encode the basic structure of the models and the tree shape of R.

We assume that the unary predicates B (signifying the start state), U and A and the binary predicate R are not in $P \cup L$.

Structure. Partitioning of the domain.

$$\forall x : A(x) \leftrightarrow \neg U(x) \wedge (\textstyle\bigwedge_{p \in P \cup L \cup \{B\}} \forall y, \boldsymbol{x} : p(y, \boldsymbol{x}) \rightarrow A(y) \wedge U(x_1) \wedge \cdots \wedge U(x_n)) \wedge (\forall x, y : R(x, y) \rightarrow A(x) \wedge A(y))$$

Tree. The predicate R encodes a tree.

$$\forall x \exists^{\leq 1} y : A(x) \rightarrow R(y, x) \wedge \exists^{\leq 1} x : \forall y : A(y) \rightarrow (\neg R(y, x) \wedge (\mu.W.x, y.R(x, y) \rightarrow W(x, y) \wedge (\exists z : W(x, y) \wedge R(y, z) \rightarrow W(x, z)))(x, y))$$

We denote the set of foundational axioms with $\Sigma_{found} = \{$**Structure.**, **Tree.**$\}$. We now turn to the axioms that encode the behavior of the production system. We omit explanations of axioms that are simply extensions of the propositional case.

Root. $B(y) \wedge \bigwedge_{r \in L} \forall \boldsymbol{x} : \neg r(y, x)$

RApp. $(\bigwedge_{r \in L} \forall \boldsymbol{x} : r(y, \boldsymbol{x}) \rightarrow \psi_r(\boldsymbol{x}))$

Appl. $(\bigwedge_{r \in L} \forall \boldsymbol{x} : \exists w (R(y, w) \wedge r_i(w, \boldsymbol{x})) \rightarrow \phi_r(y, \boldsymbol{x}) \wedge \neg \psi_r(y, \boldsymbol{x}))$

Frame. $\bigwedge_{p \in P} \forall x_1, \dots, x_n([p(y, x_1, \dots, x_n) \rightarrow (\forall w : R(y, w) \rightarrow p(w, x_1, \dots, x_n) \vee (\bigvee_{r \in L. \psi_r(\boldsymbol{z}) = \dots \neg p(t_1, \dots, t_n) \wedge \dots} \exists \boldsymbol{z} : r(y, \boldsymbol{z}) \wedge x_1 = t_1 \wedge \cdots \wedge x_n = t_n))] \wedge [\neg p(y, x_1, \dots, x_n) \rightarrow (\forall w : R(y, w) \rightarrow \neg p(w, x_1, \dots, x_n) \vee (\bigvee_{r \in L. \psi_r(\boldsymbol{z}) = \dots p(t_1, \dots, t_n) \wedge \dots} \exists \boldsymbol{z}. r(y, \boldsymbol{z}) \wedge x_1 = t_1 \wedge \cdots \wedge x_n = t_n))])$

NoFirable. $(\bigwedge_{r \in L} \forall \boldsymbol{x} : \phi_r(y, \boldsymbol{x}) \rightarrow \psi_r(y, \boldsymbol{x})) \wedge (\forall w : \neg R(y, w)))$

Firable. $(\bigvee_{i=1}^{n} \exists \boldsymbol{x} : \phi_{r_i}(y, \boldsymbol{x})) \wedge \exists w : R(y, w)$

Complete. If a rule is fireable, it is applied once.
$\bigwedge_{r \in L} \forall \boldsymbol{x} (\phi_r(y, \boldsymbol{x}) \wedge \neg \psi_r(y, \boldsymbol{x}) \rightarrow \exists^{=1} w (R(y, w) \wedge r(w, \boldsymbol{x})))$

1Rule. $(\bigvee_{r \in L} \exists \boldsymbol{x} : r(y, \boldsymbol{x})) \wedge (\bigwedge_{r \in L} (\exists \boldsymbol{z} : r(y, \boldsymbol{z}) \rightarrow \neg \bigvee_{r' \in L \& r' \neq r} \exists \boldsymbol{x} : r'(y, \boldsymbol{x})))$

WM. $\bigwedge_{p \in P} \forall x_1 \dots x_n (p(y, x_1 \dots x_n) \leftrightarrow \bigvee \{x_1 = c_1 \wedge \cdots \wedge x_n = c_n \mid p(c_1, \dots, c_n) \in WM_0\})$

Only. A rule can not be applied twice in the same state.
$(\bigwedge_{r \in L} \forall \boldsymbol{x} : r(y, \boldsymbol{x}) \rightarrow \exists^{=1} \boldsymbol{z} : (r(y, \boldsymbol{z}))$

Intermediate $=$ **RApp.** $\wedge$ **1Rule.** $\wedge$ **Only.** $\wedge$ **Appl.** $\wedge$ **Frame.** $\wedge$ **Firable.** $\wedge$ **Complete.** $\wedge$ $\neg B(y)$
End $=$ **RApp.** $\wedge$ **1Rule.** $\wedge$ **Only.** $\wedge$ **Frame.** $\wedge$ **NoFirable.** $\wedge$ $\neg B(y)$

Analogous to the propositional case, we defined a formula that captures the behavior of PS:

$$\Phi_{PS} = (\exists y : (\textbf{Root.} \wedge \textbf{NoFirable.}) \vee (\textbf{Root.} \wedge \textbf{Appl.} \wedge \textbf{Complete.} \wedge \textbf{Firable.} \wedge \\ \forall w(R(y,w) \rightarrow (\nu.X.y.(\textbf{Intermediate} \vee \textbf{End}) \wedge \forall w(R(y,w) \rightarrow X(w)))(w))))$$

The most notable difference with the propositional axiomatization is in the **Complete.** axiom. 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 between computation trees and Kripke models: they are essentially isomorphic.

Definition 5. *Let PS be an production system, WM_0 the working memory, $\mathcal{M}$ a model of Σ_{found}, and $CT^{PS}_{WM_0} = (T, V)$ the computation tree of (PS, WM_0). Then, we say that $CT^{PS}_{WM_0}$ and $\mathcal{M} = (\Delta, \cdot^{\mathcal{M}})$ are isomorphic if there is a bijective function $f : V \mapsto A^{\mathcal{M}}$ such that:*

1. *$x.i \in T^{CT}$ iff $(f(x), f(x.i)) \in (R)^{\mathcal{M}}$,*
2. *for every $x \in T$, every n-ary $p \in P \cup L$, and every $\mathbf{c} \in C^n$, $p(c_1 \dots c_n) \in V(x)$ iff $(f(x), c_1 \dots c_n) \in p^{\mathcal{M}}$,*
3. *$(z, w) \in (R)^{\mathcal{M}}$ iff $f^{-1}(w).f^{-1}(z) \in T^{CT}$, and*
4. *for every $x \in A^{\mathcal{M}}$, every n-ary $p \in P \cup L$, and every $\mathbf{c} \in C^n$, $(x, c_1 \dots c_n) \in p^{\mathcal{M}}$ iff $p(c_1 \dots c_n) \in V(f^{-1}(x))$.*

Theorem 3. *Given an FO production system $PS = (\tau, L, R)$, a starting working memory WM_0, and the formula Φ_{PS},*

1. *a model $\mathcal{M}$ of Σ_{found} is a model of Φ_{PS} iff there is a working memory WM for PS s.t. $\mathcal{M}$ is isomorphic to CT^{PS}_{WM}, and*
2. *a model $\mathcal{M}$ of Σ_{found} is a model of $\Phi_{PS} \wedge$ **WM.** iff $\mathcal{M}$ is isomorphic to $CT^{PS}_{WM_0}$.*

Proof (Sketch). We start with 2. We construct a mapping f; one can verify that it satisfies conditions 1–4 from Definition 5.

We take $y_0 \in C$ s.t. **Root.**$(y_0) \wedge$ **WM.**(y_0) holds. By **WM.** we know that $0 \in T$ and $y_0^{\mathcal{M}} \in A^{\mathcal{M}}$ "share" the same predicates. Therefore we can define $f(0) = y_0$. We proceed by induction.

For every $x.i$ in $CT^{PS}_{WM_0}$, s.t. $f(x) = y$, (recall that x is a predecessor of $x.i$ by definition) for some $y \in A^{\mathcal{M}}$, the node x and the state y share the same predicates in the sense of definition 5. We have that $r(\mathbf{c}) \in V(x.i)$ for some $r \in L$. We define $f(x.i) = z$, where $(y, z) \in R^{\mathcal{M}}$ and $(z, \mathbf{c}) \in r^{\mathcal{M}}$. There is such a unique z, by satisfaction of **Complete.**, **Firable.**, and **Appl.**. This establishes satisfaction of condition 1. Satisfaction of condition 3 is established analogously.

Satisfaction of conditions 2 and 4 is established by induction and satisfaction of **1Rule.** and **Only.** (for $p \in L$) and by satisfaction of **RApp.** and **Frame.** (for $p \in P$). Satisfaction of condition 4 is established analogously. The first part of the theorem is proved analogously.

The following result follows immediately from the undecidability of first-order logic and the fact that ϕ_r is an arbitrary first-order formula.

Theorem 4. *The satisfiability problem for ϕ_{PS} under Σ_{found} is undecidable.*

Using Theorem 3 it is straightforward to verify that finiteness of all runs can be reduced to checking entailment of

$$\forall y : A(y) \rightarrow (\mu.X.\forall w(R(y,w) \rightarrow X(w)))(y)$$

and confluence can be reduced to checking entailment of

$$\bigwedge_{p\in P} (\exists y, \boldsymbol{x} : A(y) \wedge (\forall z : A(z) \rightarrow \neg R(y,z)) \wedge p(y,\boldsymbol{x}) \rightarrow (\forall w : A(w) \wedge (\forall z : A(z) \rightarrow \neg R(w,z)) \rightarrow p(w,\boldsymbol{x}))$$

Even in the very expressive logics we consider in this paper, there are properties that might be of interest, but cannot be expressed. For example: every run of the system has the same length. This particular property cannot be expressed in FPL, or even in monadic second-order logic over countable trees [17], for that matter.

6 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 have illustrated the versatility of our approach by encoding a number of properties discussed in the literature [12,14], as well as a number of other properties that have not been previously considered. Another possible application of our encodings is the optimization of production systems. We have already shown how one can check that a particular rule is never applied (cf. property **PE6**), and thus may be discarded. Deciding equivalence of production systems can be reduced to entailment in μ-calculus and FPL. Equivalence can be exploited for optimization by replacing a production system with an equivalent system that is potentially easier to execute.

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. Finally, we plan to investigate the combination of production systems with languages for describing background knowledge, in the form of description logic ontologies.

Acknowledgements. We thank Sergio Tessaris and Enrico Franconi for valuable discussions and the anonymous reviewers for useful comments and feedback. The work presented in this paper was partially supported by the European Commission under the project ONTORULE (IST-2009-231875).

References

1. Kozen, D.: Results on the propositional μ-calculus. In: Proceedings of the 9th Colloquium on Automata, Languages and Programming, London, UK, pp. 348–359. Springer, Heidelberg (1982)
2. Gurevich, Y., Shelah, S.: Fixed-point extensions of first-order logic. In: Symposium on Foundations of Computer Science, pp. 346–353 (1985)
3. Forgy, C.: Rete: A fast algorithm for the many patterns/many objects match problem. Artif. Intell. 19(1), 17–37 (1982)
4. Grädel, E.: Guarded fixed point logics and the monadic theory of countable trees. Theor. Comput. Sci. 288(1), 129–152 (2002)
5. McCarthy, J., Hayes, P.: Some philosophical problems from the standpoint of artificial intelligence. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence, vol. 4, pp. 463–502. Edinburgh University press, Edinburgh (1969)
6. Baral, C., Lobo, J.: Characterizing production systems using logic programming and situation calculus, `http://www.public.asu.edu/~cbaral/papers/char-prod-systems.ps`
7. Kautz, H., Selman, B.: Planning as satisfiability. In: ECAI 1992: Proceedings of the 10th European Conference on Artificial Intelligence, New York, NY, USA, pp. 359–363. John Wiley & Sons, Inc., Chichester (1992)
8. Reiter, R.: Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press, Cambridge (2001)
9. Mattmller, R., Rintanen, J.: Planning for temporally extended goals as propositional satisfiability. In: Veloso, M. (ed.) Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 2007, pp. 1966–1971. AAAI Press, Menlo Park (2007)
10. Cerrito, S., Mayer, M.C.: Using linear temporal logic to model and solve planning problems. In: Giunchiglia, F. (ed.) AIMSA 1998. LNCS (LNAI), vol. 1480, p. 141. Springer, Heidelberg (1998)
11. De Giacomo, G., Lenzerini, M.: Pdl-based framework for reasoning about actions. In: AI*IA 1995: Proceedings of the 4th Congress of the Italian Association for Artificial Intelligence on Topics in Artificial Intelligence, London, UK, pp. 103–114. Springer, Heidelberg (1995)
12. Aiken, A., Hellerstein, J.M., Widom, J.: Static analysis techniques for predicting the behavior of active database rules. ACM Transactions on Database Systems 20, 3–41 (1995)
13. Baralis, E., Ceri, S., Paraboschi, S.: Compile-time and runtime analysis of active behaviors. IEEE Trans. on Knowl. and Data Eng. 10(3), 353–370 (1998)
14. Baralis, E., Torino, P.D., Widom, J., Widom, N.J.: An algebraic approach to static analysis of active database rules. ACM TODS 25, 269–332 (2000)
15. Bailey, J., Dong, G., Ramamohanarao, K.: Decidability and undecidability results for the termination problem of active database rules. In: Proceedings of the 17th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 264–273 (1998)
16. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
17. Courcelle, B.: The expression of graph properties and graph transformations in monadic second-order logic, pp. 313–400. World Scientific, Singapore (1997)