A Logic Based Approach to Static Analysis of Production Systems

to appear in [RR'09]

Martín Rezk
rezk@inf.unibz.it

KRDB Research Center
Free University of Bozen-Bolzano

Joint work with Jos de Bruijn
debruijn@inf.unibz.it

Introduction

- Research object: Production Systems (PS)
- PS consist of a set of rules: "if $condition_r$ then $action_r$"
- PS are one of the oldest knowledge representation paradigms in artificial intelligence
- Still used today to preserve consistency in databases, AI planning, semantic web etc

Introduction

- Rules (or productions) consist of two parts:
  - A sensory precondition (or "IF" statement)
  - and an action (or "THEN").
- A production system also contains a database, sometimes called working memory, which maintains data about current state of knowledge.

Introduction

Given a working memory, the rule interpreter applies rules in three steps:

1. *pattern matching*,
2. *conflict resolution*, and
3. *rule execution*

The Problem

- Rule-based systems are administered and executed in a distributed environment
- 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.

The Goals

- 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,
  - μ-Calculus
  - Fixed-point logic (FPL)

Formalization of Propositional Case

Propositional Production Systems

Formalization of Propositional Case

Formalization

Formal description of any language is a prerequisite to any rigorous methods of proof, validation, or verification.

A *Generic Production System PS*: is defined as a tuple

$$PS = (Prop, L, R)$$

Where

- *Prop* is a finite set of proposition, representing the set of potential facts.
- *L* is a set of n rule labels appearing in R .
- *R* is a set of n rules, which are statements of the form

$$r_k : \text{ if } \phi_k \text{ then } add(r_k) \ / \ remove(r_k)$$

Formalization of Propositional Case

where

- r_k is an element from L, and $0 < k < n + 1$
- ϕ_k is a propositional formula (the precondition of r_k)
- $add : L \rightarrow 2^{Prop}$
- $remove : L \rightarrow 2^{Prop}$

Formalization - Definitions

- Definition: Working memory
- Definition: *Concrete Production System* (PS, WM_0)
- Definition: *Application* of a rule
  $WM_j = WM_i \cup add(r_k) \setminus remove(r_k)$
- Definition: A *run* $WM_0 \longrightarrow^{r_{k_1}} WM_1 \longrightarrow^{r_{k_{n-1}}} \dots \longrightarrow^{r_{k_n}} WM_n \dots$

Formalization - Def

A *computation tree* $CT^{PS}_{WM_0}$ for a CPS, is a $(P \cup L) - labeled$ tree (T,V) where

- The root of T is 0
- $V : T \rightarrow 2^{Prop}$
- $V(0) = WM_0$
- 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 $CT^{PS}_{WM_0}$.

Formalization - Example

Example

- $Prop = \{o, q, w, z\}$ And the rules:
- $r_1 =$ if $\bigwedge_{p \in Prop} \neg p$ then o
- $r_2 =$ if $\bigwedge_{p \in Prop} \neg p$ then q
- $r_3 =$ if $\bigwedge_{p \in Prop} \neg p$ then w
- $r_4 =$ if $\bigwedge_{p \in Prop} \neg p$ then z
- $r_5 =$ if q then z
- $r_6 =$ if q then w
- $WM_0 = \emptyset$

This production system has six rules, four of them can be fired in the initial working memory, producing a new working memory:

Formalization - Example

Example

$$WM_0 \rightarrow_1^r WM_1 \text{ where } WM_1 = \{o\}$$

$$WM_0 \rightarrow_2^r WM_1' \text{ where } WM_1' = \{q\}$$

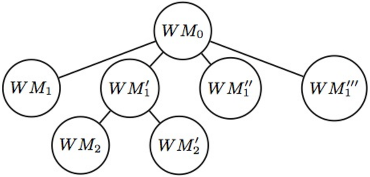
$$WM_0 \rightarrow_3^r WM_1'' \text{ where } WM_1'' = \{w\}$$

$$WM_0 \rightarrow_4^r WM_1''' \text{ where } WM_1''' = \{z\}$$

$$WM_1' \rightarrow_5^r WM_1''' \text{ where } WM_1''' = \{qz\}$$

$$WM_1'' \rightarrow_6^r WM_1''' \text{ where } WM_1''' = \{qw\}$$

Formalization - Example



Formalization - Axiomatization

μ-Calculus

- *Var* : (infinite) set of variable names,
- *Prop*: set of atomic propositions,
- μ-Calculus extends propositional logic with the modal operator $\Diamond$ and a fix point operator μ
- We obtain 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 $\neg\mu.Z.\neg\phi(\neg Z)$.

Formalization - Axiomatization

- 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,
  - $V : S \rightarrow 2^{Prop}$ 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.

Formalization - Axiomatization

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
- $(\mu.Z.\phi(Z))^K(\mathcal{V}) = \bigcap\{S' \subseteq S \mid \phi^K(\mathcal{V}[Z \leftarrow S']) \subseteq S'\}$.

Formalization - Axiomatization

Root The current state represents the root of the CT.

RApp Rule application: If rule is applied then the action holds. In this point we also add some kind of strategy which is usually required, and it is that if a rule does not change the working memory, then it can not be applied.

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

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.

NoFireable No rule is fireable and there is no successor.

Formalization - Axiomatization

Fireable	At least one rule is fireable and there is a successor.
Complete	If a rule is fireable, it is applied in some successor states.
1Rule	Exactly one rule is applied.
WM	(Optional) The initial working memory holds.

Formalization - Axiomatization

$$\text{Intermediate} = \mathbf{RApp} \wedge \mathbf{1Rule} \wedge \mathbf{Appl} \wedge \mathbf{Frame} \wedge \mathbf{Fireable} \wedge \mathbf{Complete} \wedge \neg b$$

$$\text{End} = \mathbf{RApp} \wedge \mathbf{1Rule} \wedge \mathbf{Frame} \wedge \mathbf{NoFireable} \wedge \neg b$$

Formalization - Axiomatization

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

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

Formalization - Bisimulation

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 for all $(s, t) \in Z$,

– $(s_0, t'_0) \in Z$

–For every $(s_i, t'_i) \in Z$, $p \in V(s_i)$ if and only if $p \in V'(t'_i)$, 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 $(s', t') \in Z$, and

– 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 $(s', t') \in Z$, and

Formalization - Theorem

Theorem

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

Formalization - Therorems

Summary

$$
\begin{tikzcd}
\mathit{ProdSys} \arrow[r] \arrow[d] & \mathit{Axioms} \arrow[d] \\
\mathit{CompTree} \arrow[r] & \mathit{Kripke}
\end{tikzcd}
$$

Formalization - Properties

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

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

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

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

Formalization - Therorems

Theorem

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

Formalization - Complexity

Theorem

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

Formalization of First Order Case

First Order Production Systems

Formalization of First Order Case

- Fixed Point Logics FPL extends standard first order logic with least fixed-point formulas of the form $[\mu W.\vec{x}.\psi(W, \vec{x})](\vec{x})$,
- In order to obtain the necessary correspondence with the constants employed in the production system, we assume *standard names*.

Formalization of First Order Case

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 $\psi(W, \vec{x})$ defines an operator on k-ary relations $W \subseteq A^K$:

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

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

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

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

Formalization - FO

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

$$r: \text{ if } \phi_r(\vec{x}) \text{ then } \psi_r(\vec{x})$$

Formalization - FO

where :
– $r \in L$,
– ϕ_r is an FO formula with free variables $\vec{x}$ and
– $\psi_r(\vec{x}) = (a_1 \wedge \cdots \wedge a_k \wedge \neg b_1 \wedge \cdots \wedge \neg b_l)$, where
$a_1, \ldots, a_k, b_1, \ldots, b_l$ are atomic formulas with free variables among $\vec{x}$, 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
–$\phi_r^{remove} = \{b_1, \ldots, b_l\}$.

Formalization - FO

The *grounding* of an FO production system $PS = (\tau, L, R)$, denoted $gr(PS)$, is obtained from PS by replacing each rule

$$r: \text{ if } \phi_r(\vec{x}) \text{ then } \psi_r(\vec{x})$$

with a set of rules $\mathcal{S}(r(\vec{x})): \text{ if } \mathcal{S}(\phi_r(\vec{x})) \text{ then } \mathcal{S}(\psi_r(\vec{x}))$, for every substitution $\mathcal{S}$of variables with constants in C.

Formalization - FO

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.

Theorem

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

[a]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.

Formalization - FO

When considering concrete FO production systems, we can also exploit grounding, provided the conditions in the rules are *domain-independent*

- 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).

Formalization - FO

Theorem

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 $\phi_r(\vec{x})$ is domain-independent, and let WMbe a working memory. Then, the properties **PE1-7** *can be decided in double exponential time, on (PS, WM).*

Formalization - Axioms

First Order Axiomatization

- We capture the structure of the computation tree using the binary predicate R, and a set of foundational axioms.
- 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.

Formalization - Axioms

Foundational axioms:

Structure Partitioning of the domain.

Tree The predicate R encodes a tree.

We denote the set of foundational axioms with $\Sigma_{found} = \{\textbf{Structure}, \textbf{Tree}\}$.

Formalization - Axioms

We extend the previous set of axioms to the first order case. The most relevant changes are:

Complete If a rule is fireable, it is applied once.

Only A rule can not be applied twice in the same state.

Formalization - Axioms

$$\begin{aligned}\text{Intermediate} &= \mathbf{RApp} \wedge \mathbf{1Rule} \wedge \mathbf{Only} \wedge \mathbf{Appl} \wedge \mathbf{Frame} \wedge \\ &\quad \mathbf{Fireable} \wedge \mathbf{Complete} \wedge \neg B(y) \\ \text{End} &= \mathbf{RApp} \wedge \mathbf{1Rule} \wedge \mathbf{Only} \wedge \mathbf{Frame} \wedge \mathbf{NoFireable} \wedge \neg B(y)\end{aligned}$$

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

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

Formalization - Axioms

Theorem

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 $CT^{PS}_{WM_0}$.*

Formalization - Axioms

Theorem

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

Related Work

We consider two streams of related work:

- *action languages and planning* and
- *rules in active databases*.

Related Work

- Situation Calculus (John McCarthy and Patrick Hayes 1969): A distinguishing feature between situation calculus and the logics used here, is the notion of situation and the notion of state
  - Arguably, the latter are conceptually a better match with the notion of working memory in production systems
- In (Chitta Baral and Jorge Lobo), they use logic programs with the stable model semantics and situation calculus notation for characterizing production systems
  - We allow the initial working memory to vary

Related Work

Planning - STRIPS

- Robert Mattmller and Jussi Rintanen (2007) 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
    - We present properties which can not be expressed in LTL
- 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

Related Work

- De Giacomo Giuseppe and Lenzerini Maurizio (1995) 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.

Related Work

- 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, and a formal proof of the correspondence with the set of runs of a PS, and the models of our axiomatization. This link is required to do formal verification of properties of PS, using the models of the axiomatization.

Related Work

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
- Second, we extend the propositional case, and we model PS with variables, First Order Production Systems (*FO-PS*) using FPL. The choice of μ-calculus makes the path from the propositional PS to FO-PS more understandable.

Related Work

Rules in active databases are strongly related to production rules

- The works of (A. Aiken and J M. Hellerstein and J. Widom 1995; Baralis E. and Ceri S. and Paraboschi S. 1998; Elena Baralis and Politecnico Di Torino and Jennifer Widom and Name Jennifer Widom 2000) are based on checking properties of graphs
- The general problem, where conditions are arbitrary SQL queries, is (unsurprisingly) known to be undecidable
- Elena Baralis and Politecnico Di Torino and Jennifer Widom and Name Jennifer Widom (2000) 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

Conclusion

- We presented an embedding of P-PS into μ-calculus, and FO-PS 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
- We have illustrated the versatility of our approach

Future Work

- We plan to extend both P-PS and FO-PS 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
- We plan to look for new decidable fragments of our first-order encoding
- We plan to investigate the combination of production systems with languages for describing background knowledge, in the form of description logic ontologies.

Some philosophical problems from the standpoint of artificial intelligence

Characterizing Production Systems using Logic Programming and Situation Calculus

Static analysis techniques for predicting the behavior of active database rules

An Algebraic Approach to Static Analysis of Active Database Rules

Planning for temporally extended goals as propositional satisfiability

PDL-based framework for reasoning about actions

Compile-Time and Runtime Analysis of Active Behaviors

Thx

Thanks!

Thx

Thanks!