

Predicate Abstraction of Rewrite Theories

Kyungmin Bae and José Meseguer

Department of Computer Science,
University of Illinois at Urbana-Champaign, Urbana IL 61801
{kbae4,meseguer}@cs.uiuc.edu

Abstract. For an infinite-state concurrent system $\mathcal{S}$ with a set AP of state predicates, its *predicate abstraction* defines a finite-state system whose states are subsets of AP , and its transitions $s \rightarrow s'$ are witnessed by concrete transitions between states in $\mathcal{S}$ satisfying the respective sets of predicates s and s' . Since it is not always possible to find such witnesses, an over-approximation adding extra transitions is often used. For systems $\mathcal{S}$ described by formal specifications, predicate abstractions are typically built using various automated deduction techniques. This paper presents a new method—based on rewriting, semantic unification, and variant narrowing—to automatically generate a predicate abstraction when the formal specification of $\mathcal{S}$ is given by a conditional rewrite theory. The method is illustrated with concrete examples showing that it naturally supports abstraction refinement and is quite accurate, i.e., it can produce abstractions not needing over-approximations.

1 Introduction

To automatically verify a temporal logic property φ of an infinite-state system $\mathcal{S}$ by model checking, two methods can be used: (i) we can try to use an *infinite-state* model checking procedure to *directly* verify that $\mathcal{S} \models \varphi$, or (ii) we can *abstract* $\mathcal{S}$ into a *finite-state* system $\mathcal{A}$, verify that $\mathcal{A} \models \varphi$ holds using a standard model checking method, and use general abstraction results to *prove* that this ensures $\mathcal{S} \models \varphi$ for the original system $\mathcal{S}$. These two methods have complementary strengths. On the one hand, a direct, infinite-state model checking method can settle the question once and for all, but may not always terminate, and may only be applicable under some restrictions on $\mathcal{S}$ and φ . On the other hand, abstractions allow the use of efficient finite-state model checking algorithms and tools, but may exhibit *spurious counterexamples*; that is, counterexamples showing that $\mathcal{A} \not\models \varphi$, while in fact $\mathcal{S} \models \varphi$ holds. In such cases, *abstraction refinement* [6] can be used to find a less abstract, yet still finite, abstraction $\mathcal{A}'$ where we may show $\mathcal{A}' \models \varphi$ if indeed φ holds for the system $\mathcal{S}$.

Predicate abstraction is one of the simplest and most widely used abstraction methods. We assume that several *state predicates* $AP = \{p_1, \dots, p_n\}$ are defined on the states of $\mathcal{S}$. Then, the abstract system $\mathcal{S}/AP$ has set of states $\mathcal{P}(AP)$, and a transition $s \rightarrow s'$ exists for $s, s' \in \mathcal{P}(AP)$ iff there exists a concrete transition $u \rightarrow v$ in $\mathcal{S}$ such that u (resp. v) satisfies exactly the predicates in s (resp. in

s'). Since it may not be always possible to prove the existence of such a concrete transition $u \rightarrow v$, an *over-approximation* $\alpha(\mathcal{S}/AP)$, which adds extra transitions when in doubt, may instead be used. Predicate abstraction can be used either for: (i) systems $\mathcal{S}$ defined by *programs* in a conventional language like C, or (ii) systems $\mathcal{S}$ described by *formal specifications*, in which case some theorem proving techniques are used to build a predicate abstraction $\alpha(\mathcal{S}/AP)$.

This paper presents an automatic method to build a predicate abstraction $\alpha(\mathcal{R}/AP)$ for a concurrent system specified as a *rewrite theory* $\mathcal{R} = (\Sigma, E, R)$ with equations E and rewrite rules R . We systematically exploit rewriting and unification techniques to *automate* the predicate abstraction process by a method that, as we illustrate with examples, has a good chance of building abstractions that are as *accurate* as possible. What is exploited is *both* the simplicity of the specification as a rewrite theory $\mathcal{R}$, and the power and automatic nature of rewriting and unification techniques, including the recently developed *generic* method to automatically derive unification algorithms by *variant narrowing* [14], already available in Maude. Although several abstraction methods have been studied before for rewrite theories (for a detailed survey see § 3.12 in [20]), except for the technical report [23] based on theorem proving, predicate abstraction of rewrite theories has remained undeveloped until now.

Our contributions. Our main contributions are as follows:

- *Automatic method*: unlike the quite different semi-automatic method in [23], which generated proof obligations for an interactive theorem prover, our method is fully automatic, although it may produce an over-approximation.
- *Wide applicability beyond narrowing-based methods*: narrowing-based model checking techniques (e.g., in [4,13]) require equations E to have a finitary E -unification algorithm and rules R to be unconditional; no such restrictions apply to $\mathcal{R} = (\Sigma, E, R)$ in our abstraction method; and
- *Effective unsatisfiability checking procedure*, which, although incomplete, is automatic and can be used in practice to prove a conjunction of equations E -unsatisfiable. We believe this method can be useful not just for building predicate abstractions, but for other formal verification tasks.

Related Work. Predicate abstraction was first introduced in [16], and has been widely applied to both conventional programming languages, e.g., [5,25], and formal specifications, e.g., [10,17]. However, predicate abstraction for rewrite theories has *not* been much developed, except for the semi-automatic method in [23] using theorem proving. This work is also related to existing rewriting-based methods for model checking infinite-state systems, for example, using equational abstractions [21], narrowing [4,13], or tree automata [15,22]. The present work differs from and complements those approaches in that predicate abstraction always reduces such an infinite-state system to a finite-state one. Finally, our method to check unsatisfiability of equational constraints is related to the techniques in [2,12] to check unfeasibility of conditional critical pairs in the context of proving confluence of conditional rewrite rules.

2 Preliminaries

Equational Theories. An order-sorted signature is a triple $\Sigma = (S, \leq, \Sigma)$ with poset of sorts $(S, \leq)$ and operators $\Sigma = \{\Sigma_{w,s}\}_{(w,s) \in S^* \times S}$ typed in $(S, \leq)$. A Σ -algebra A consists of a set A_s for each sort s , a subset inclusion $A_s \subseteq A_{s'}$ for each subsort relation $s \leq s'$, and a function $A_f : A_{s_1} \times \dots \times A_{s_n} \rightarrow A_s$ for each operator $f \in \Sigma_{s_1 \dots s_n, s}$. The set $\mathcal{T}_\Sigma(\mathcal{X})_s$ denotes the set of Σ -terms of sort s over $\mathcal{X}$ an infinite set of S -sorted variables, and $\mathcal{T}_{\Sigma,s}$ denotes the set of ground Σ -terms of sort s . We assume that $\mathcal{T}_{\Sigma,s} \neq \emptyset$ for each sort s in Σ . A *substitution* $\sigma : Y \rightarrow \mathcal{T}_\Sigma(\mathcal{X})$ is a function that maps variables $Y \subseteq \mathcal{X}$ to terms of the same sort, and is homomorphically extended to $\mathcal{T}_\Sigma(\mathcal{X})$ in a natural way.

A Σ -equation is an unoriented pair $t = t'$ of terms $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})_s$ for some sort $s \in \Sigma$. For a Σ -algebra A and a valuation $a : \mathcal{X} \rightarrow A$ assigning a value in A_s to each variable $x \in \mathcal{X}$ of sort s , if $\bar{a} : \mathcal{T}_\Sigma(\mathcal{X}) \rightarrow A$ is the homomorphic extension of a to terms, by definition, $(A, a) \models t = t'$ iff $\bar{a}(t) = \bar{a}(t')$. An *equational theory* (Σ, E) with a set of Σ -equations E induces a congruence relation $=_E$ such that $u =_E u'$ iff $(\Sigma, E) \vdash (\forall \mathcal{X}) u = u'$ iff $(A, a) \models u = u'$ for any model A of E (i.e., for each equation $t = t' \in E$ and valuation a , $(A, a) \models t = t'$ holds) [18].

An E -unifier of an equation $t = t'$ is a substitution σ such that $\sigma t =_E \sigma t'$. A set $CSU_E(t = t')$ is a *complete set of E -unifiers* in which any E -unifier ρ of $t = t'$ has a more general E -unifier $\sigma \in CSU_E(t = t')$, i.e., $(\exists \eta) \rho =_E \eta \circ \sigma$. For a set of equations E and a set of equational axioms B (such as associativity, commutativity, and identity), if $E \cup B$ has the *finite variant property*, there is a finitary $E \cup B$ -unification algorithm to find *finite $CSU_{E \cup B}(t = t')$* [9,14]. As explained in [9], $E \cup B$ has the finite variant property iff for every term t there is a bound n such that the normal form of θt for a normalized substitution θ is reachable from t by applying E modulo B less than n times.

Rewrite Theories. A rewrite theory is a formal specification of a concurrent system [19]. We consider *order-sorted rewrite theories* $\mathcal{R} = (\Sigma, E, R)$ with (Σ, E) an equational theory and R a set of *rewrite rules* of the form $l \longrightarrow r \text{ if } C$, where $l, r \in \mathcal{T}_\Sigma(\mathcal{X})_s$ for some sort s and C is a conjunction of Σ -equations. The system's states are modeled as the initial algebra $\mathcal{T}_{\Sigma/E}$ (i.e., each state is an E -equivalence class $[t]_E \in \mathcal{T}_{\Sigma/E}$ of ground terms). Each rule $(l \longrightarrow r \text{ if } C) \in R$ defines a *one-step rewrite* $t \longrightarrow_{R,E} t'$, specifying the system's transitions, iff there exist a subterm u of t and a substitution σ such that $u =_E \sigma l$ and t' is the term obtained from t by replacing u by σr , provided that $\sigma u =_E \sigma v$ for each condition $u = v$ in C .

A rewrite theory $\mathcal{R}$ is called *topmost* iff there exists a sort **State** at the top of one of the connected component of $(S, \leq)$ such that: (i) for each rewrite rule $l \longrightarrow r \text{ if } C$, both l and r have the top sort **State**; and (ii) no operator in Σ has **State** or any of its subsorts as an argument sort. This ensures that all rewrites with rules in R must take place at the top of the term. Throughout this paper we assume that E decomposes as a disjoint union $E = E_o \cup B$ with B a set of equational axioms (such as associativity, commutativity, and identity) and E_o *convergent* (i.e., sort-decreasing, terminating, confluent, and coherent modulo B) [11], and that R is coherent with E_o module B [24].

LTL Model Checking. A linear temporal logic (LTL) formula is constructed by state propositions $p \in AP$ and temporal logic operators such as $\neg$ (negation), $\wedge$, $\vee$, $\rightarrow$ (implication), $\Box$ (always), $\Diamond$ (eventually), $\mathbf{U}$ (until), and $\mathbf{O}$ (next). The semantics of LTL is defined on a *Kripke structure* $\mathcal{K} = (S, AP, \mathcal{L}, \rightarrow_{\mathcal{K}})$ [7], where S is a set of *states*, AP is a set of *state propositions*, $\mathcal{L} : S \rightarrow \mathcal{P}(AP)$ is a *state-labeling function*, and $\rightarrow_{\mathcal{K}} \subseteq S \times S$ is a *transition relation*.

Given two Kripke structures $\mathcal{K}_i = (S_i, AP, \mathcal{L}_i, \rightarrow_{\mathcal{K}_i})$, $i = 1, 2$, a relation $H \subseteq S_1 \times S_2$ is a *simulation* iff (i) if $s_1 H s_2$, then $\mathcal{L}_1(s_1) = \mathcal{L}_2(s_2)$, and (ii) if $s_1 H s_2$ and $s_1 \rightarrow_{\mathcal{K}_1} s'_1$, there exists $s'_2 \in S_2$ such that $s'_1 H s'_2$ and $s_2 \rightarrow_{\mathcal{K}_2} s'_2$. A simulation $H \subseteq S_1 \times S_2$ is *total* iff for each $s_1 \in S_1$ there exists $s_2 \in S_2$ such that $s_1 H s_2$, and H is a *bisimulation* iff both H and H^{-1} are simulations.

Lemma 1. [7] *Given a simulation H between two Kripke structures $\mathcal{K}_1$ and $\mathcal{K}_2$, if $s_0^1 H s_0^2$, then for any LTL formula φ , $\mathcal{K}_2, s_0^2 \models \varphi$ implies $\mathcal{K}_1, s_0^1 \models \varphi$.*

We can associate to a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ a corresponding Kripke structure. A state proposition p is defined as a term of sort **Prop**, whose meaning is defined by equations using the auxiliary operator $_ \models _ : \text{State Prop} \rightarrow \text{Bool}$. By definition, $p \in \mathcal{T}_{\Sigma/E, \text{Prop}}$ is satisfied on $[t]_E$ iff $(t \models p) =_E \text{true}$. We assume that: (i) sort **Bool** has two constants *true* and *false* with $\text{true} \neq_E \text{false}$; (ii) any $t \in \mathcal{T}_{\Sigma, \text{Bool}}$ is provably equal to either *true* or *false*; and (iii) $\mathcal{R}$ is deadlock-free, since $\mathcal{R}$ can be easily transformed into an equivalent deadlock-free theory [21].

Definition 1. *Given $\mathcal{R} = (\Sigma, E, R)$ and a set $AP \subseteq \mathcal{T}_{\Sigma/E, \text{Prop}}$ defined by E , the corresponding Kripke structure is $\mathcal{K}(\mathcal{R})_{AP} = (\mathcal{T}_{\Sigma/E, \text{State}}, AP, \mathcal{L}_{AP}, \rightarrow_{R, E})$, where $\mathcal{L}_{AP}([t]_E) = \{p \in AP \mid (t \models p) =_E \text{true}\}$.*

3 AP-Abstractions of Rewrite Theories

This section shows how a predicate abstraction of a topmost rewrite theory $\mathcal{R} = (\Sigma, E, R)$ can be constructed by using E -unification. Following the usual predicate abstraction methods, we consider abstract states as subsets of a set of state propositions $AP = \{p_1, \dots, p_n\}$, where a transition $s \rightarrow s' \in \mathcal{P}(AP)^2$ is defined if there exists a *concrete* one-step rewrite $t \rightarrow_{R, E} t'$ such that:

$$s = \{p \in AP \mid (t \models p) =_E \text{true}\} \quad \wedge \quad s' = \{p \in AP \mid (t' \models p) =_E \text{true}\} \quad (1)$$

Our approach is motivated by the following observation. For a topmost rewrite theory $\mathcal{R} = (\Sigma, E, R)$, a *concrete* one-step rewrite $t \rightarrow_{R, E} t'$ exists iff for a rule $(l \rightarrow r \text{ if } C) \in R$ and a *ground* substitution σ :

$$t =_E \sigma l \quad \wedge \quad t' =_E \sigma r \quad \wedge \quad (\forall u = v \in C) \sigma u =_E \sigma v \quad (2)$$

The abstraction $s \rightarrow s'$ of the transition $t \rightarrow_{R, E} t'$ is given by Condition (1). Since $t =_E \sigma l$ and $t' =_E \sigma r$ by Condition (2), we can replace t and t' in (1) by σl and σr , respectively, and we obtain: $s = \{p \in AP \mid (\sigma l \models p) =_E \text{true}\}$, $s' = \{p \in AP \mid (\sigma r \models p) =_E \text{true}\}$, and $s \rightarrow s' \text{ if } (\forall u = v \in C) \sigma u =_E \sigma v$. That is, $s \rightarrow s'$ holds if there exist a rewrite rule $(l \rightarrow r \text{ if } C) \in R$ and a ground substitution σ that satisfy these E -equality constraints.

Abstract Kripke Structures. We construct an AP -abstract Kripke structure of $\mathcal{R}$, whose states are elements of $\mathcal{P}(AP)$, and whose transitions are decided by solving E -equality constraints. Let $p \in_- : \mathcal{P}(AP) \rightarrow \{true, false\}$ be the truth function such that $(p \in s) = true$ iff $p \in s$, and $(p \in s) = false$ iff $p \notin s$.

Definition 2. Given a topmost rewrite theory $\mathcal{R} = (\Sigma, E, R)$ and a finite set of state propositions AP defined by E , the AP -abstract Kripke structure of $\mathcal{R}$ is a Kripke structure $\mathcal{R}/AP = (\mathcal{P}(AP), AP, id_{\mathcal{P}(AP)}, \longrightarrow_{\mathcal{R}/AP})$, where:

- $id_{\mathcal{P}(AP)} : \mathcal{P}(AP) \rightarrow \mathcal{P}(AP)$ is the identity labeling function, and
- $s \longrightarrow_{\mathcal{R}/AP} s'$ iff there exists a rewrite rule $(l \longrightarrow r \text{ if } C) \in R$ such that the following constraints are E -satisfiable:

$$\bigwedge_{p \in AP} (l \models p) = (p \in s) \wedge \bigwedge_{p \in AP} (r \models p) = (p \in s') \wedge \bigwedge_{u=v \in C} u = v \quad (\dagger)$$

If there exists a finitary E -unification algorithm (e.g., E has the finite variant property), the satisfiability of $(\dagger)$ can be decided by checking for the emptiness of the *finite* complete set of the E -unifiers:

$$CSU_E(\bigwedge_{p \in AP} (l \models p) = (p \in s) \wedge \bigwedge_{p \in AP} (r \models p) = (p \in s') \wedge \bigwedge_{u=v \in C} u = v).$$

However, checking satisfiability of the constraints $(\dagger)$ by E -unification is in general undecidable [3]. Therefore, $\mathcal{R}/AP$ may *not* have an effective procedure to *precisely* decide its transitions. In practice, there are three cases:

1. For *some* rule in R , we can prove the *satisfiability* of $(\dagger)$ for states s and s' , in which case we know that $s \longrightarrow_{\mathcal{R}/AP} s'$ holds.
2. For *every* rule in R , we can prove the *unsatisfiability* of $(\dagger)$ for states s and s' , in which case we know that $s \longrightarrow_{\mathcal{R}/AP} s'$ does *not* hold.
3. Otherwise, we cannot decide whether $s \longrightarrow_{\mathcal{R}/AP} s'$ holds or not. In this case we can *add* $s \longrightarrow_{\alpha(\mathcal{R}/AP)} s'$ to a Kripke structure $\alpha(\mathcal{R}/AP)$ *approximating* (and therefore *simulating*) $\mathcal{R}/AP$ with possibly more transitions.

By definition, a Kripke structure $\alpha(\mathcal{K}) = (S, AP, \mathcal{L}, \longrightarrow_{\alpha(\mathcal{K})})$ is an *approximation* of $\mathcal{K} = (S, AP, \mathcal{L}, \longrightarrow_{\mathcal{K}})$ iff $\longrightarrow_{\mathcal{K}} \subseteq \longrightarrow_{\alpha(\mathcal{K})}$. In Section 4 we propose some procedures for checking satisfiability of the constraints $(\dagger)$.

Theorem 1. For an LTL formula φ and a state $t \in \mathcal{T}_{\Sigma, \text{State}}$, if $s = \mathcal{L}_{AP}([t]_E)$, then $\alpha(\mathcal{R}/AP), s \models \varphi$ implies $\mathcal{K}(\mathcal{R})_{AP}, [t]_E \models \varphi$.

Proof. It suffices to show that the state-labeling function $\mathcal{L}_{AP}$ of $\mathcal{K}(\mathcal{R})_{AP}$, where $\mathcal{L}_{AP}([u]_E) = \{p \in AP \mid (u \models p) =_E true\}$, is a total simulation from $\mathcal{K}(\mathcal{R})_{AP}$ to $\mathcal{R}/AP$. Then, $(id_{\mathcal{P}(AP)} \circ \mathcal{L}_{AP}) = \mathcal{L}_{AP}$ becomes a simulation from $\mathcal{K}(\mathcal{R})_{AP}$ to $\alpha(\mathcal{R}/AP)$, since $id_{\mathcal{P}(AP)}$ is a simulation from $\mathcal{R}/AP$ to $\alpha(\mathcal{R}/AP)$.

Suppose that $t \longrightarrow_{R, E} t'$ exists in $\mathcal{K}(\mathcal{R})_{AP}$. By definition, for a rewrite rule $(l \longrightarrow r \text{ if } C) \in R$, there is a ground substitution σ such that $t =_E \sigma l$, $t' =_E \sigma r$, and for each $u = v \in C$, $\sigma u =_E \sigma v$. Let $s = \mathcal{L}_{AP}([t]_E)$ and $s' = \mathcal{L}_{AP}([t']_E)$. Since $t =_E \sigma l$ and $t' =_E \sigma r$, we have $s = \{p \in AP \mid (\sigma l \models p) =_E true\}$ and $s' = \{p \in AP \mid (\sigma r \models p) =_E true\}$. Hence, $\bigwedge_{p \in AP} (\sigma l \models p) =_E (p \in s)$ and $\bigwedge_{p \in AP} (\sigma r \models p) =_E (p \in s')$ hold. That is, σ is a solution of $(\dagger)$, so that $s \longrightarrow_{\mathcal{R}/AP} s'$. Therefore, $\mathcal{L}_{AP}$ is a total simulation from $\mathcal{K}(\mathcal{R})_{AP}$ to $\mathcal{R}/AP$. $\square$

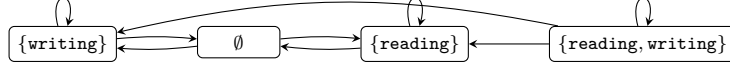


Fig. 1. The AP -abstract Kripke structure with the initial state $\emptyset$.

Rule	$\langle 0, 0 \rangle \rightarrow \langle 0, s(0) \rangle$	$\langle r, 0 \rangle \rightarrow \langle s(r), 0 \rangle$	$\langle r, s(w) \rangle \rightarrow \langle r, w \rangle$	$\langle s(r), w \rangle \rightarrow \langle r, w \rangle$
$\emptyset \rightarrow \{w\}$	$\langle 0, 0 \rangle \rightarrow \langle 0, s(0) \rangle$			
$\emptyset \rightarrow \{r\}$		$\langle 0, 0 \rangle \rightarrow \langle s(0), 0 \rangle$		
$\{w\} \rightarrow \emptyset$			$\langle 0, s(0) \rangle \rightarrow \langle 0, 0 \rangle$	
$\{w\} \rightarrow \{w\}$			$\langle 0, s(s(w)) \rangle \rightarrow \langle 0, s(w) \rangle$	
$\{r\} \rightarrow \emptyset$				$\langle s(0), 0 \rangle \rightarrow \langle 0, 0 \rangle$
$\{r\} \rightarrow \{r\}$		$\langle s(r), 0 \rangle \rightarrow \langle s(s(r)), 0 \rangle$		$\langle s(s(r)), 0 \rangle \rightarrow \langle s(r), 0 \rangle$
$\{r, w\} \rightarrow \{w\}$				$\langle s(0), s(w) \rangle \rightarrow \langle 0, s(w) \rangle$
$\{r, w\} \rightarrow \{r\}$			$\langle s(r), s(0) \rangle \rightarrow \langle s(r), 0 \rangle$	
$\{r, w\} \rightarrow \{r, w\}$			$\langle s(r), s(s(w)) \rangle \rightarrow \langle s(r), s(w) \rangle$	$\langle s(s(r)), s(w) \rangle \rightarrow \langle s(r), s(w) \rangle$

Fig. 2. The rule instances for each abstract transition $s \rightarrow s' \in \mathcal{P}(AP)^2$.

Example. We illustrate our ideas with a simple model of the readers-writers problem (adapted from [8]). Each state is modeled as a pair $\langle r, w \rangle \in \mathbb{N}^2$ in which r is the number of readers and w is the number of writers, given by the operator $\langle _, _ \rangle : \text{Nat Nat} \rightarrow \text{State}$. Natural numbers are expressed in Peano notation using the successor function $s : \text{Nat} \rightarrow \text{Nat}$ and the zero constant 0 of sort Nat . The behavior is defined by the following four unconditional rewrite rules: $\langle 0, 0 \rangle \rightarrow \langle 0, s(0) \rangle$, $\langle r, 0 \rangle \rightarrow \langle s(r), 0 \rangle$, $\langle r, s(w) \rangle \rightarrow \langle r, w \rangle$, and $\langle s(r), w \rangle \rightarrow \langle r, w \rangle$. Mutual exclusion is expressed by the LTL formula $\Box \neg(\text{reading} \wedge \text{writing})$, where the state propositions are defined by the equations: $\langle s(r), w \rangle \models \text{reading} = \text{true}$, $\langle 0, w \rangle \models \text{reading} = \text{false}$, $\langle r, s(w) \rangle \models \text{writing} = \text{true}$, and $\langle r, 0 \rangle \models \text{writing} = \text{false}$, satisfying the finite variant property since the right-hand sides are constants.

This system is infinite-state, since the number of readers r is unbounded. However, we obtain the *finite* AP -abstract Kripke structure $\mathcal{R}/AP$ in Figure 1, whose transitions are decided by using E -unification. Figure 2 shows the rule instances $\zeta l \rightarrow \zeta r$ for each rule $l \rightarrow r \in R$, transition $s \rightarrow s' \in \mathcal{P}(AP)^2$, and E -unifier $\zeta \in CSU_E(\bigwedge_{p \in AP} (l \models p) = (p \in s) \wedge \bigwedge_{p \in AP} (r \models p) = (p \in s'))$ that represent the transitions of $\mathcal{R}/AP$. The property $\Box \neg(\text{reading} \wedge \text{writing})$ holds from the initial state $\emptyset$ in $\mathcal{R}/AP$, since $\{\text{reading}, \text{writing}\}$ is *not* reachable. Therefore, $\Box \neg(\text{reading} \wedge \text{writing})$ also holds from $\langle 0, 0 \rangle$ in $\mathcal{K}(\mathcal{R})_{AP}$, thanks to Theorem 1.

If we consider the LTL formula $\Box \Diamond \neg \text{writing}$ (i.e., infinitely often not writing), there exists the spurious counterexample $\emptyset \rightarrow \{\text{writing}\} \rightarrow \{\text{writing}\} \rightarrow \dots$. As usual for predicate abstraction methods, we can *refine* the AP -abstraction by adding the state proposition $1w$ to further specialize the abstract state space, meaning that there exists only one writer, defined by the three equations: $\langle R, s(0) \rangle \models 1w = \text{true}$, $\langle R, 0 \rangle \models 1w = \text{false}$, and $\langle R, s(s(w)) \rangle \models 1w = \text{false}$. We have then the refined AP -abstract Kripke structure in Figure 3 in which the formula $\Box \Diamond \neg \text{writing}$ holds from the initial state $\emptyset$. Again, by Theorem 1, $\Box \Diamond \neg \text{writing}$ also holds from $\langle 0, 0 \rangle$ in the concrete Kripke structure $\mathcal{K}(\mathcal{R})_{AP}$.

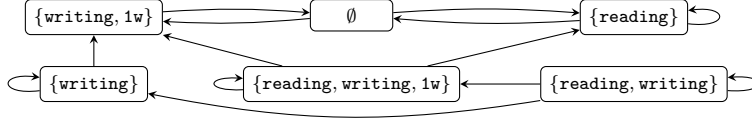


Fig. 3. Refined AP -abstract Kripke structure for the readers-writers problem.

4 Effective Procedures for Equality Constraints

In Section 3 we defined a predicate abstraction $\mathcal{R}/AP$ of a topmost rewrite theory $\mathcal{R} = (\Sigma, E, R)$ using E -equality constraints when the set of state propositions AP is specified by E . If there exists a finitary E -unification algorithm, such as E satisfying the finite variant property, then we can automatically construct $\mathcal{R}/AP$ by using E -unification. This section shows how to check the *unsatisfiability* of E -equality constraints. Since the problem is in general semi-decidable [3], we are interested in a *sound but incomplete* terminating procedures that can be automated, so that an over-approximation $\alpha(\mathcal{R}/AP)$ can be built when the procedure fails to give an answer. A semi-decidable E -unification algorithm does not give such a procedure because it may *not* terminate when no solution exists.

Our method relies on the fact that a state proposition p is typically defined using only a subset of equations $E_p \subseteq E$, so that solving constraints for p may only involve E_p , *not* all of E . If a finitary E_p -unification algorithm is available, then we can discharge the constraints for p using E_p -unification. After resolving all such *solvable* constraints, we apply a sound procedure based on E -reduction to test if the remaining constraint are inconsistent or not.

Decomposition of Constraints. In practice, the equational semantics of a state proposition p can be restricted to a subset of the equations $E_p \subseteq E$. As assumed throughout this paper, E decomposes as a disjoint union $E = E_o \cup B$ with B a set of equational axioms and E_o convergent modulo B .

Definition 3. Let $\Omega \subseteq \Sigma$ be a set of free constructors for E_o modulo B . Given a set of patterns $U = \{u_1, \dots, u_n\} \subseteq \mathcal{T}_\Sigma(\mathcal{X})$, we define:

$$\begin{aligned} \llbracket U \rrbracket &= \{t \in \mathcal{T}_\Sigma(\mathcal{X}) \mid (\exists \sigma : \mathcal{X} \rightarrow \mathcal{T}_\Omega(\mathcal{X}), \exists u \in U) t =_B \sigma u\}, \\ E_U &= \{u = v \in E_o \mid u, v \in \llbracket U \rrbracket\}. \end{aligned}$$

We call E_o syntactically independent with respect to U iff E_U is convergent modulo B and satisfies: (i) any proper subterm v of $u \in U$ is strongly E_o, B -irreducible (i.e., γv is a normal form for any normalized substitution γ), and (ii) for every $t = t' \in E - E_U$ and $u \in U$, $CSU_B(t = u) = \emptyset$.

For the readers-writers problem in Section 3, every state proposition p has its syntactically independent patterns U_p ; e.g., for the state proposition *reading*, for $U_{\text{reading}} = \{(S \models \text{reading}), \text{true}, \text{false}\}$ for a variable S of sort **State**, where $E_{U_{\text{reading}}} = \{\langle s(r), w \rangle \models \text{reading} = \text{true}, \langle 0, w \rangle \models \text{reading} = \text{false}\}$.

We are interested in finding a subset of equations $G \subseteq E$ with a finitary unification algorithm that can make E -solvability of a constraint $u = v$ in $(\dagger)$ decidable by using G -unification. An E -equality constraint $u = v$ is G -solvable for a subset of equations $G \subseteq E$ iff $\sigma u =_E \sigma v$ implies $\sigma u =_G \sigma v$ for any substitution σ . Since $G \subseteq E$, if $u = v$ is G -solvable, then $\sigma u =_E \sigma v \iff \sigma u =_G \sigma v$.

Proposition 1. *If E_o is syntactically independent with respect to U , then for any $u, v \in \llbracket U \rrbracket$, an E -equality constraint $u = v$ is $E_U \cup B$ -solvable.*

Proof. Since we assume that $\mathcal{T}_\Sigma(\mathcal{X})_s \neq \emptyset$ for each sort s , a constraint $u = v$ is $E_o \cup B$ -solvable iff for some normalized ground substitution $\sigma : \mathcal{X} \rightarrow \mathcal{T}_\Omega$, $\text{can}_{E_o/B}(\sigma u) =_B \text{can}_{E_o/B}(\sigma v)$, where $\text{can}_{E_o/B}(t)$ denotes an E_o/B -canonical form of t . Since $\sigma u, \sigma v \in \llbracket U \rrbracket$, by Ω -terms being free modulo B , convergence of E_U modulo B , and (i)–(ii) in Definition 3, $\text{can}_{E_o/B}(\sigma u) =_B \text{can}_{E_U/B}(\sigma u)$ and $\text{can}_{E_o/B}(\sigma v) =_B \text{can}_{E_U/B}(\sigma v)$ hold. The lifting lemma for narrowing modulo B then forces $u = v$ to be $E_U \cup B$ -solvable by narrowing with E_U modulo B . $\square$

Even when $E_U \cup B$ does *not* have the finite variant property, there may still exist a subset $\tilde{E}_U \subseteq E_U$ where $\tilde{E}_U \cup B$ has the finite variant property and a constraint $u = v$ is $\tilde{E}_U \cup B$ -solvable (see Section 5 for an example).

For a set of E -equality constraints D , if $G \subseteq E$ has a finitary G -unification algorithm and a constraint $u = v \in D$ is G -solvable, then $CSU_G(u = v)$ is finite, and $u = v$ is E -satisfiable iff $CSU_G(u = v) \neq \emptyset$. Therefore, we can decompose the problem D into finding $\zeta \in CSU_G(u = v)$ and solving the remaining constraints.

Lemma 2. *Given a set of E -equality constraints D , if $G \subseteq E$ has a finitary unification algorithm and $u = v \in D$ is G -solvable, then D is E -satisfiable iff for some $\zeta \in CSU_G(u = v)$, $\{\zeta u = \zeta v \mid u = v \in D - \{u = v\}\}$ is E -satisfiable*

Proof. Suppose that there exists a substitution ρ such that $\bigwedge_{u=v \in D} \rho u =_E \rho v$. Since $u = v$ is G -solvable, $\rho u =_{E_{FV}} \rho v$ also holds. Therefore, there exists a substitution $\zeta \in CSU_{E_{FV}}(u = v)$ such that $(\exists \eta) \rho =_{E_{FV}} \eta \circ \zeta$. Notice that $\rho =_E \eta \circ \zeta$, since $G \subseteq E$. Therefore, $\bigwedge_{u=v \in D - \{u=v\}} \eta(\zeta u) =_E \eta(\zeta v)$ holds. $\square$

We can repeatedly apply this procedure to solve each G -solvable constraint in $(\dagger)$ to determine $s \rightarrow_{\mathcal{R}/AP} s'$, provided that G has a finitary unification algorithm. If there exists *no* common solution, then $s \rightarrow_{\mathcal{R}/AP} s'$ does *not* hold.

Unfeasibility of Constraints. Applying Lemma 2, we can transform a set D of E -equality constraints into an equivalent set F having *no* G -solvable constraints. We now present a sound but incomplete procedure to test for E -unsatisfiability of F . We have assumed that E decomposes as a disjoint union $E = E_o \cup B$, where E_o is convergent and a finitary B -unification algorithm exists, and $\mathcal{T}_{\Sigma,s} \neq \emptyset$ for each sort $s \in \Sigma$. Therefore, we can use the *canonical term algebra* $\mathcal{C}_{\Sigma/E_o,B}$, whose elements are B -equivalence classes of Σ -terms in E_o/B -canonical form, and which is an initial $E_o \cup B$ -algebra. That is, F is E -satisfiable iff there exists a normalized ground substitution θ such that $(\mathcal{C}_{\Sigma/E_o,B}, q_B \circ \theta) \models F$, where $q_B : \mathcal{T}_\Sigma \rightarrow \mathcal{T}_{\Sigma/B}$ is the quotient map $t \mapsto [t]_B$ for B .

Given the set X of the variables in F , let $\bar{X} = \{\bar{x} \mid x \in X\}$ be the set in which each variable $x \in X$ of sort s is turned into the constant $\bar{x}$ of the same sort s , where $\bar{X} \cap \Sigma = \emptyset$, and let $\bar{F} = \{\bar{u} = \bar{v} \mid u = v \in F\}$ be the set of the *ground* constraints obtained from F by replacing each $x \in X$ by $\bar{x} \in \bar{X}$. Recall that a $\Sigma \cup \bar{X}$ -algebra is exactly a pair (A, a) with a valuation $a : \bar{X} \rightarrow A$. Therefore, if $(\mathcal{C}_{\Sigma/E_o, B}, q_B \circ \theta) \models F$, then the valuation $q_B \circ \hat{\theta}$, with $\hat{\theta}(\bar{x}) = \theta(x)$ for $x \in X$, gives us a $\Sigma \cup \bar{X}$ -algebra $(\mathcal{C}_{\Sigma/E_o, B}, q_B \circ \hat{\theta})$ satisfying both E and $\bar{F}$. Soundness of equational logic then ensures that whenever $(\Sigma \cup \bar{X}, E \cup \bar{F}) \vdash (\forall \emptyset) \bar{u} = \bar{v}$, where $\text{vars}(u) \cup \text{vars}(v) \subseteq X$, we must have $(\mathcal{C}_{\Sigma/E_o, B}, q_B \circ \hat{\theta}) \models (\forall \emptyset) \bar{u} = \bar{v}$.

Our sound procedure for testing unsatisfiability of F is based on the idea of obtaining a proof of the form $(\Sigma \cup \bar{X}, E \cup \bar{F}) \vdash (\forall \emptyset) \bar{u} = \bar{v}$, where u and v are *strongly E_o, B -irreducible Σ -terms* (i.e., each normalized substitution γ makes both γu and γv normalized) and $CSU_B(u = v) = \emptyset$. This gives a contradiction to the assumption that F is satisfiable; if F is satisfied by a normalized substitution θ , then $(\mathcal{C}_{\Sigma/E_o, B}, q_B \circ \hat{\theta}) \models \bar{u} = \bar{v}$, i.e., $[\theta u]_B = [\theta v]_B$, since u and v are strongly E_o, B -irreducible, but $CSU_B(u = v) = \emptyset$ implies $[\theta u]_B \neq [\theta v]_B$.

A practical way of obtaining such a proof $(\Sigma \cup \bar{X}, E \cup \bar{F}) \vdash (\forall \emptyset) \bar{u} = \bar{v}$ is by rewriting modulo B . That is, we can use the set of rewrite rules $E_o^\rightarrow \cup \bar{F}^\rightarrow$ with $E_o^\rightarrow$ the oriented equations and for a B -compatible order $\succ$ on ground terms:

$$\begin{aligned} \bar{F}^\rightarrow = \{ & \text{can}_{E_o/B}(\bar{w}) \rightarrow \text{can}_{E_o/B}(\bar{w}') \mid w = w' \in F \text{ or } w' = w \in F, \\ & \text{can}_{E_o/B}(\bar{w}) \succ \text{can}_{E_o/B}(\bar{w}') \}, \end{aligned}$$

where $\text{can}_{E_o/B}(w)$ denotes an E_o/B -canonical form of w . Then, if we obtain $\bar{u} \xrightarrow{*}_{E_o^\rightarrow \cup \bar{F}^\rightarrow, B} \bar{v}$ by rewriting modulo B , then by equational reasoning, we have a *fortiori* derived $(\Sigma \cup \bar{X}, E \cup \bar{F}) \vdash (\forall \emptyset) \bar{u} = \bar{v}$. In summary:

Theorem 2. *For a set of E -equality constraints F with $X = \text{vars}(F)$, if there exist strongly E_o, B -irreducible Σ -terms u and v such that $\text{vars}(u) \cup \text{vars}(v) \subseteq X$, $CSU_B(u = v) = \emptyset$, and $\bar{u} \xrightarrow{*}_{E_o^\rightarrow \cup \bar{F}^\rightarrow, B} \bar{v}$, then F is E -unsatisfiable.*

Example. We consider a model of the readers-writers problem with explicit shared variables and processes, adapted from [1] (which is a more sophisticated version of the model in Section 3). Each state has the form $\langle n, b \mid p_1; \dots; p_n \rangle$, given by the operator $\langle _, _ \mid _ \rangle : \text{Nat} \text{ Bool ProcSet} \rightarrow \text{State}$, where n denotes the number of readers, b is a Boolean flag to denote *no* readers and *no* writers, and $p_1; \dots; p_n$ is a multiset of processes, each in a status $p_i \in \{\text{idle}, \text{read}, \text{write}\}$. The behavior of the system is specified by the following (conditional) rewrite rules:

$$\begin{aligned} \langle n, b \mid \text{idle} ; PS \rangle &\longrightarrow \langle s(n), b' \mid \text{read} ; PS \rangle && \text{if } c(n, b, b') = \text{true} \\ \langle s(n), b \mid \text{read} ; PS \rangle &\longrightarrow \langle n, b' \mid \text{idle} ; PS \rangle && \text{if } c(n, b', b) = \text{true} \\ \langle n, \text{true} \mid \text{idle} ; PS \rangle &\longrightarrow \langle n, \text{false} \mid \text{write} ; PS \rangle \\ \langle n, \text{false} \mid \text{write} ; PS \rangle &\longrightarrow \langle n, \text{true} \mid \text{idle} ; PS \rangle \end{aligned}$$

where the function $c(n, b, b')$ returns *true* iff $b = \text{true}$ and $b' = \text{false}$ when $n = 0$, or $b = b'$ when $n > 0$, defined by the equations:

$$\begin{aligned}
c(0, true, false) &= true & c(0, false, b) &= false & c(0, b, true) &= false \\
c(s(n), b, b) &= true & c(s(n), true, false) &= false & c(s(n), false, true) &= false
\end{aligned}$$

Mutual exclusion is expressed by the formula $\Box \neg (reading \wedge writing)$. We can define sorts **ProcIdleSet** (denoting processes in *idle*), **ProcReadSet** (for processes in *idle* or *read*), and **ProcWriteSet** (denoting processes in *idle* or *write*), having subsort relations $\text{ProcIdleSet} \leq \text{ProcReadSet}$ $\text{ProcWriteSet} \leq \text{ProcSet}$. The state propositions are then defined by the following equations, where RS is a variable of sort **ProcReadSet** and WS is a variable of sort **ProcWriteSet**:

$$\begin{aligned}
\langle n, b \mid read ; PS \rangle &\models reading = true & \langle n, b \mid WS \rangle &\models reading = false \\
\langle n, b \mid write ; PS \rangle &\models writing = true & \langle n, b \mid RS \rangle &\models writing = false
\end{aligned}$$

This system is finite-state for a *fixed* set of processes; but the system is actually infinite-state, since the number of processes is unbounded. We are interested in verifying $\Box \neg (reading \wedge writing)$ for an unbounded number of processes.

In order to verify $\Box \neg (reading \wedge writing)$, we need to have two additional state propositions *free* and *good*, defined by the following equations, where $good(n, PS)$ returns *true* iff n is equal to the number of readers in PS :

$$\begin{aligned}
\langle n, true \mid PS \rangle &\models free = true & \langle n, false \mid PS \rangle &\models free = false \\
\langle n, b \mid PS \rangle &\models good = good(n, PS) & good(s(n), read ; PS) &= good(n, PS) \\
good(n, write ; PS) &= good(n, PS) & good(n, idle ; PS) &= good(n, PS) \\
good(0, WS) &= true & good(s(n), WS) &= false & good(0, read ; PS) &= false
\end{aligned}$$

Notice that every state proposition has syntactically independent equations. Hence, we can easily see that every constraint for c , *reading*, *writing*, and *free* is solvable by a set of equations satisfying the finite variant property. However, the equations defining *good* do *not* have the finite variant property.

We can obtain a finite $\mathcal{R}/AP$ in which only two states $\{reading, good\}$ and $\{writing, good\}$ are reachable from the initial state $\{free, good\}$. For example, from $\{free, good\}$, after resolving each $E_p \cup B$ -solvable constraint, we have the following sets of the remaining constraints:

$$\begin{aligned}
\langle 0, true \mid idle ; IS \rangle &\models good = true \wedge \langle s(0), false \mid read ; IS \rangle \models good = b_1, \\
\langle s(n), true \mid idle ; IS \rangle &\models good = true \wedge \langle s(s(n)), true \mid read ; IS \rangle \models good = b_2, \\
\langle n, true \mid idle ; IS \rangle &\models good = true \wedge \langle n, false \mid write ; IS \rangle \models good = b_3,
\end{aligned}$$

where IS has sort **ProcIdleSet** and $b_1, b_2, b_3 \in \{true, false\}$. By normalizing each constraint after replacing the variables into the constants, we have the oriented constraint sets $\{true \rightarrow true, true \rightarrow b_1\}$, $\{true \rightarrow false, b_2 \rightarrow false\}$, and $\{good(\bar{n}, \bar{IS}) \rightarrow true, good(\bar{n}, \bar{IS}) \rightarrow b_3\}$. Therefore, the cases of $b_1 = false$, $b_2 \in \{true, false\}$, and $b_3 = false$ are unsatisfiable. That is, $\{free, good\}$ has only two next states $\{reading, good\}$ and $\{writing, good\}$. Similarly, $\{reading, good\}$ has the next states $\{free, good\}$ and $\{reading, good\}$, and $\{writing, good\}$ has the next states $\{free, good\}$ and $\{writing, good\}$. Therefore, $\Box \neg (reading \wedge writing)$ holds in $\mathcal{R}/AP$ from $\{free, good\}$. Thanks to Theorem 1, the formula also holds in $\mathcal{K}(\mathcal{R})_{AP}$ for an unbounded number of processes.

```

findNextStates( $s \in \mathcal{P}(AP)$ ):
  NextStates :=  $\emptyset$ ;
  foreach  $s' \in \mathcal{P}(AP)$  do
    foreach ( $l \rightarrow r$  if  $C$ )  $\in R$  do
      CTR := getConstraints( $s, s', l, r, C$ ); // CTR = the set of constraints ( $\dagger$ )
      (D,F,G) := findSolvable(CTR); // D = a set of G-solvable equations
      if isSatisfiable(D,F,G) then add  $s'$  to NextStates;
  return NextStates;

isSatisfiable(D, F, G):
  if  $D \neq \emptyset$  then
    choose  $u = v$  from D;
    foreach  $\zeta \in CSUG(u = v)$  do
      if isSatisfiable( $\zeta(D - \{u = v\}), \zeta F, G$ ) then return true;
    return false;
  else
    return (if  $F = \emptyset$  then true else  $\neg$  testUnsatisfiable(F));

```

Fig. 4. Predicate Abstraction Algorithm for $\mathcal{R} = (\Sigma, E, R)$.

Predicate Abstraction Algorithm. Figure 4 shows an algorithm to generate a predicate abstraction $\alpha(\mathcal{R}/AP)$ of a rewrite theory $\mathcal{R} = (\Sigma, E, R)$. There exists a transition $s \rightarrow s'$ in $\alpha(\mathcal{R}/AP)$ iff $s' \in \text{findNextStates}(s)$, where the function findNextStates(s) returns a set of next *abstract* states from $s \in \mathcal{P}(AP)$.

The function findNextStates uses a number of subroutines that correspond to methods in Sections 3 and 4. Given a set of E -equality constraints CTR , using Proposition 1, findSolvable(CTR) returns a triple (D, F, G) such that D is a set of G -solvable constraints, G has the finite variant property, and $CTR = D \cup F$. Then, using Lemma 2, isSatisfiable(D, F, G) returns *false* if the set of constraints $D \cup F$ is unsatisfiable, where testUnsatisfiable(F) returns *true* if F is shown to be unsatisfiable using Theorem 2. More details on each subroutine, as well as some demos based on several components of this algorithm, can be found at <http://formal.cs.illinois.edu/kbae/pred>.

5 Case Study

We illustrate our predicate abstraction method with another nontrivial infinite-state system, namely, Lamport's bakery protocol for mutual exclusion (adapted from [4,13]). Each state has the form $\langle n, m, A, [i_1, d_1] \dots [i_n, d_n] \rangle$, given by the operator $\langle _, _, _, _ \rangle : \text{Nat Nat Action ProcSet} \rightarrow \text{State}$, where n is the current number in the bakery's number dispenser, m is the number currently being served, $A \in \{\perp, \text{wake}, \text{crit}, \text{exit}\}$ is the action taken in the previous step, and $[i_1, d_1] \dots [i_n, d_n]$ is a multiset of customer processes, each with an identity i_j and in a mode d_j . A mode can be *idle* (not yet picked a number), *wait*(n) (waiting with number n), or *crit*(n) (being served with number n).

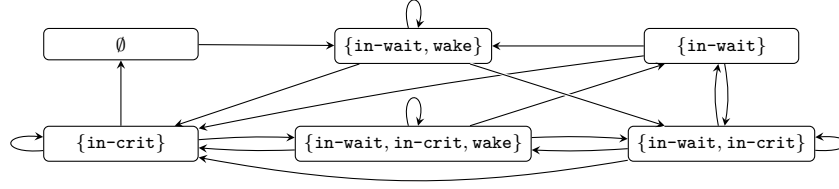


Fig. 5. *AP*-abstract Kripke structure for the livelock freedom property.

The behavior is then specified by the following unconditional rewrite rules, where natural numbers are modeled as multisets of s with the multiset union operator $_ : \text{Nat Nat} \rightarrow \text{Nat}$ (empty syntax), satisfying laws of commutativity and associativity, and the empty multiset 0 (e.g., $0 = 0$, and $3 = s s s$):

$$\begin{aligned}
\langle n, m, A, [i, \text{idle}] PS \rangle &\rightarrow \langle s n, m, \text{wake}, [i, \text{wait}(n)] PS \rangle \\
\langle n, m, A, [i, \text{wait}(m)] PS \rangle &\rightarrow \langle n, m, \text{crit}, [i, \text{crit}(m)] PS \rangle \\
\langle n, m, A, [i, \text{crit}(m)] PS \rangle &\rightarrow \langle n, s m, \text{exit}, [i, \text{idle}] PS \rangle
\end{aligned}$$

Notice that this system is infinite-state since: (i) the counters n and m are unbounded, and (ii) the number of customer processes is unbounded.

We can define three sorts **ProcIdleSet** (for processes in *idle*), **ProcWaitSet** (denoting processes in *idle* or *wait*(n)), and **ProcCritSet** (denoting processes in *idle* or *crit*(n)), yielding an order-sorted signature with subsort relations $\text{ProcIdleSet} \leq \text{ProcWaitSet}$ $\text{ProcCritSet} \leq \text{ProcSet}$. We can define sort **NWAction** denoting the *non-wake* actions ($\perp$, *crit*, and *exit*), where $\text{NWAction} \leq \text{Action}$.

Livelock Freedom. We are interested in verifying the livelock freedom “*if some process is waiting, then some (possibly different) process eventually enters the critical section*” under the fairness assumption “*wake is not taken infinitely many times*,” expressed as the LTL formula $\Box \Diamond \neg \text{wake} \rightarrow \Box (\text{in-wait} \rightarrow \Diamond \text{in-crit})$. The state propositions are defined by the following equations that satisfy the finite variant property, where CS is a variable of sort **ProcCritSet**, WS is a variable of sort **ProcWaitSet**, and NA is a variable of sort **NWAction**:

$$\begin{aligned}
\langle n, m, A, [i, \text{wait}(k)] PS \rangle \models \text{in-wait} = \text{true} \quad &\langle n, m, A, CS \rangle \models \text{in-wait} = \text{false} \\
\langle n, m, A, [i, \text{crit}(k)] PS \rangle \models \text{in-crit} = \text{true} \quad &\langle n, m, A, WS \rangle \models \text{in-crit} = \text{false} \\
\langle n, m, \text{wake}, PS \rangle \models \text{wake} = \text{true} \quad &\langle n, m, NA, PS \rangle \models \text{wake} = \text{false}.
\end{aligned}$$

We obtain the *AP*-abstract Kripke structure in Figure 5 using *E*-unification, because a finitary *E*-unification algorithm is available. Since *in-crit* holds in every state in the second row, the formula $\Box \Diamond \neg \text{wake} \rightarrow \Box (\text{in-wait} \rightarrow \Diamond \text{in-crit})$ holds in the *AP*-abstract Kripke structure from the initial state $\emptyset$. Hence, by Theorem 1, the formula also holds in the concrete Kripke structure $\mathcal{K}(\mathcal{R})_{AP}$ from any state $[t]_E$ with $\mathcal{L}_{AP}([t]_E) = \emptyset$ for an unbounded number of processes.

Mutual Exclusion. We now consider the mutual exclusion “*at most one process can enter the critical section*,” expressed by the LTL formula $\Box \text{mutex}$ in which the state proposition is defined by the equations: $\langle n, m, A, WS \rangle \models \text{mutex} = \text{true}$, $\langle n, m, A, [i, \text{crit}(k)] WS \rangle \models \text{mutex} = \text{true}$, and

$$\langle n, m, A, [i, \text{crit}(k)] [i', \text{crit}(k')] PS \rangle \models \text{mutex} = \text{false}.$$

We need three extra state propositions to define a predicate abstraction: *mcrit*, *bound*, and *uniq*. First, *mcrit* holds in a state $\langle n, m, A, PS \rangle$ if at most one process enters the critical section with number m , defined by the equations: $\langle n, m, A, WS \rangle \models \text{mcrit} = \text{true}$, $\langle n, m, A, [i, \text{crit}(m)] WS \rangle \models \text{mcrit} = \text{true}$, and

$$\begin{aligned} \langle n, m, A, [i, \text{crit}(s k m)] WS \rangle &\models \text{mcrit} = \text{false} \\ \langle n, s k m, A, [i, \text{crit}(m)] WS \rangle &\models \text{mcrit} = \text{false} \\ \langle n, m, A, [i, \text{crit}(k)] [i', \text{crit}(k')] PS \rangle &\models \text{mcrit} = \text{false}. \end{aligned}$$

Next, the state proposition *bound* holds in a state $\langle n, m, A, PS \rangle$ if any ticket number of a process in *PS* is less than n , defined by the equations:

$$\left. \begin{aligned} \langle n, m, A, PS \rangle &\models \text{bound} = \text{bd}(n, PS) \\ \text{bd}(n, [i, \text{wait}(n k)] PS) &= \text{false} \quad \text{bd}(n, [i, \text{crit}(n k)] PS) = \text{false} \end{aligned} \right] (E_b)$$

$$\begin{aligned} \text{bd}(n, IS) &= \text{true} & \text{bd}(s n k, [i, \text{wait}(k)] PS) &= \text{bd}(s n k, PS) \\ \text{bd}(s n k, [i, \text{crit}(k)] PS) &= \text{bd}(s n k, PS) \end{aligned}$$

Finally, the state proposition *uniq* holds in a state $\langle n, m, A, PS \rangle$ if no duplicate ticket numbers of processes in *PS*, defined by the equations:

$$\left. \begin{aligned} \langle n, m, A, PS \rangle &\models \text{uniq} = q(PS) \quad q([i, \text{wait}(k)] [j, \text{wait}(k)] PS) = \text{false} \\ q([i, \text{wait}(k)] [j, \text{crit}(k)] PS) &= \text{false} \quad q([i, \text{crit}(k)] [j, \text{crit}(k)] PS) = \text{false} \end{aligned} \right] (E_q)$$

$$\begin{aligned} q([i, \text{wait}(k)] [j, \text{wait}(s m k)] PS) &= q([i, \text{wait}(k)] PS) & q([i, \text{idle}] PS) &= q(PS) \\ q([i, \text{wait}(k)] [j, \text{crit}(s m k)] PS) &= q([i, \text{wait}(k)] PS) & q([i, \text{wait}(k)]) &= \text{true} \\ q([i, \text{crit}(k)] [j, \text{wait}(s m k)] PS) &= q([i, \text{crit}(k)] PS) & q([i, \text{crit}(k)]) &= \text{true} \\ q([i, \text{crit}(k)] [j, \text{crit}(s m k)] PS) &= q([i, \text{crit}(k)] PS) & q(\text{none}) &= \text{true} \end{aligned}$$

Notice that every state proposition has syntactically independent equations. In particular, the equations for *mutex* and *mcrit* have the finite variant property. The equations defining *bound* and *uniq* do *not* have the finite variant property, but the equations $E_b \cup B$ and $E_q \cup B$ that define the *negative* cases of *bound* and *uniq* do have the finite variant property. Furthermore:

Lemma 3. *For a state term $u \in \mathcal{T}_\Sigma(\mathcal{X})_{\text{State}}$, a constraint $(u \models \text{bound} = \text{false})$ is $E_b \cup B$ -solvable, and $(u \models \text{uniq} = \text{false})$ is $E_q \cup B$ -solvable.*

Proof. A term u has the form $\langle n, m, A, t_{procs} \rangle$. If $\sigma u \models \text{bound} =_E \text{false}$ for some substitution σ , then σt_{procs} contains a process whose ticket number is greater than or equal to n . Since $E_b \cup B$ reduces such a negative case to *false* in 2 steps, $\sigma u \models \text{bound} =_{E_b \cup B} \text{false}$. Similarly, if $\sigma u \models \text{uniq} =_E \text{false}$, then σt_{procs} contains two processes with the same ticket number, and $\sigma u \models \text{uniq} =_{E_q \cup B} \text{false}$. $\square$

We obtain $\mathcal{R}/AP$ having a *single* reachable state from the given initial state $\{mutex, mcrit, bound, uniq\}$. By resolving each constraint for $mutex$ and $mcrit$, from $\{mutex, mcrit, bound, uniq\}$, the resulting sets of remaining constraints are $\{l_k \models bound = true, l_k \models uniq = true, r_k \models bound = b_k, r_k \models uniq = b'_k\}$ for $1 \leq j \leq 5$, where $b_k, b'_k \in \{true, false\}$ and each pair $l_k \rightarrow_k r_k$ is given by:

$$\begin{aligned} \langle n, m, A, [i, idle] \rangle WS &\rightarrow_1 \langle s\ n, m, wake, [i, wait(n)] \rangle WS \\ \langle n, m, A, [i, idle] [j, crit(m)] \rangle WS &\rightarrow_2 \langle s\ n, m, wake, [i, wait(n)] [j, crit(m)] \rangle WS \\ \langle n, m, A, [i, wait(m)] \rangle WS &\rightarrow_3 \langle n, m, crit, [i, crit(m)] \rangle WS \\ \langle n, m, A, [i, crit(m)] \rangle WS &\rightarrow_4 \langle n, s\ m, exit, [i, idle] \rangle WS \\ \langle n, m, A, [i, wait(m)] [j, crit(m)] \rangle WS &\rightarrow_5 \langle n, m, crit, [i, crit(m)] [j, crit(m)] \rangle WS \end{aligned}$$

The case of $k = 5$ is unsatisfiable for any values of $b_5, b'_5 \in \{true, false\}$, since $l_5 \models uniq =_E false$. For the cases of $1 \leq k \leq 4$, if $b_k = false$ for $bound$, then its solution $\zeta \in CSU_{E_b \cup B}(r_k \models bound = false)$ makes $\zeta l_k \models bound =_E false$.¹ That is, the cases of $b_k = false$ are unsatisfiable. Similarly, if $b'_k = false$ for $uniq$, then $\zeta \in CSU_{E_a \cup B}(r_k \models uniq = false)$ makes either $\zeta l_k \models bound =_E false$ or $\zeta l_k \models uniq =_E false$, i.e., unsatisfiable. Therefore, $\{mutex, mcrit, bound, uniq\}$ has the one next state, itself. Clearly, $\Box mutex$ holds in $\mathcal{R}/AP$, and thus $\Box mutex$ also holds in $\mathcal{K}(\mathcal{R})_{AP}$ for an unbounded number of processes.

6 Concluding Remarks

In this work we have presented a predicate abstraction method for a wide range of infinite-state concurrent systems specified as conditional rewrite theories. One of the main ideas is to reduce a predicate abstraction problem into solving a set of E -equality constraints, which can be automated if there exists a finitary E -unification algorithm. To deal with the case where no finitary E -unification algorithm exists, we have also presented sound but incomplete procedures to check unsatisfiability of E -equality constraints, where an over-approximation can be constructed if such procedures fail to give an answer. We have illustrated our method with nontrivial infinite-state systems, namely, parameterized protocols for an unbounded number of processes.

Future work includes implementing a predicate abstraction-based LTL model checker in Maude,² integrating it with other infinite-state model checking methods such as equational abstractions and narrowing-based methods, and applying SMT techniques, as well as E -unification, to solve E -equality constraints.

Acknowledgments. This work has been supported in part by NSF Grant CNS 13-19109 and AFOSR Grant FA8750-11-2-0084.

¹ For example, $CSU_{E_b \cup B}(\langle s\ n, m, wake, [j, wait(n)] \rangle WS \models bound = false)$ has the substitution $\zeta = \{WS \leftarrow [j', wait(s\ n\ m')] WS'\}$, and $\zeta l_1 \models bound =_E false$.

² The key software components used by the algorithm (already available in Maude) and detailed experiments showing how they can automate the case studies are explained in the webpage <http://formal.cs.illinois.edu/kbae/pred>.

References

1. Abdulla, P., Chen, Y.F., Delzanno, G., Haziza, F., Hong, C.D., Rezine, A.: Constrained monotonic abstraction. In: CONCUR. Springer (2010)
2. Avenhaus, J., Loria-Sáenz, C.: On conditional rewrite systems with extra variables and deterministic logic programs. In: LPAR. pp. 215–229. Springer (1994)
3. Baader, F., Snyder, W.: Unification theory. In: Handbook of Automated Reasoning, pp. 445–532. Elsevier and MIT Press (2001)
4. Bae, K., Escobar, S., Meseguer, J.: Abstract logical model checking of infinite-state systems using narrowing. In: RTA. LIPIcs, vol. 21, pp. 81–96 (2013)
5. Ball, T., Majumdar, R., Millstein, T., Rajamani, S.K.: Automatic predicate abstraction of C programs. ACM SIGPLAN Notices 36(5), 203–213 (2001)
6. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: CAV. pp. 154–169. Springer (2000)
7. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press (2001)
8. Clavel, M., Durán, F., Eker, S., Meseguer, J., Lincoln, P., Martí-Oliet, N., Talcott, C.: All About Maude, LNCS, vol. 4350. Springer (2007)
9. Comon-Lundh, H., Delaune, S.: The finite variant property: How to get rid of some algebraic properties. In: RTA, pp. 294–307. Springer (2005)
10. Das, S., Dill, D.L., Park, S.: Experience with predicate abstraction. In: CAV. pp. 160–171. Springer (1999)
11. Dershowitz, N., Jouannaud, J.P.: Rewrite systems. In: Handbook of Theoretical Computer Science, Vol. B, pp. 243–320. North-Holland (1990)
12. Durán, F., Meseguer, J.: A Church-Rosser checker tool for conditional order-sorted equational Maude specifications. In: RTA, pp. 69–85. Springer (2010)
13. Escobar, S., Meseguer, J.: Symbolic model checking of infinite-state systems using narrowing. In: RTA. pp. 153–168 (2007)
14. Escobar, S., Sasse, R., Meseguer, J.: Folding variant narrowing and optimal variant termination. J. Algebraic and Logic Programming 81, 898–928 (2012)
15. Genet, T., Rusu, V.: Equational approximations for tree automata completion. Journal of Symbolic Computation 45(5), 574–597 (2010)
16. Graf, S., Saïdi, H.: Construction of abstract state graphs with PVS. In: CAV. pp. 72–83. Springer (1997)
17. Lahiri, S.K., Bryant, R.E., Cook, B.: A symbolic approach to predicate abstraction. In: CAV. pp. 141–153. Springer (2003)
18. Meseguer, J.: Membership algebra as a logical framework for equational specification. In: WADT. LNCS, vol. 1376, pp. 18–61. Springer (1997)
19. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theor. Comp. Sci. 96(1), 73–155 (1992)
20. Meseguer, J.: Twenty years of rewriting logic. J. Algebraic and Logic Programming 81, 721–781 (2012)
21. Meseguer, J., Palomino, M., Martí-Oliet, N.: Equational abstractions. Theor. Comp. Sci. 403(2-3), 239–264 (2008)
22. Ohsaki, H., Seki, H., Takai, T.: Recognizing boolean closed A-tree languages with membership conditional rewriting mechanism. In: RTA. Springer (2003)
23. Palomino, M.: A predicate abstraction tool for maude (2005), available at <http://maude.sip.ucm.es/~miguelpt/bibliography.html>
24. Viry, P.: Equational rules for rewriting logic. Theor. Comp. Sci. 285 (2002)
25. Visser, W., Havelund, K., Brat, G., Park, S., Lerda, F.: Model checking programs. Automated Software Engineering 10(2), 203–232 (2003)