Abstract Logical Model Checking of Infinite-State Systems Using Narrowing

Kyungmin Bae[1], **Santiago Escobar**[2], **and José Meseguer**[1]

1 University of Illinois at Urbana-Champaign, IL, USA
{kbae4,meseguer}@cs.uiuc.edu

2 Universitat Politècnica de València, Spain
sescobar@dsic.upv.es

Abstract

A concurrent system can be naturally specified as a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ where states are elements of the initial algebra $\mathcal{T}_{\Sigma/E}$ and concurrent transitions are axiomatized by the rewrite rules R. Under simple conditions, narrowing with rules R modulo equations E can be used to symbolically represent the system's state space by means of terms with logical variables. We call this symbolic representation a *logical state space* and it can also be used for model checking verification of LTL properties. Since in general such a logical state space can be infinite, we propose several abstraction techniques for obtaining either an over-approximation or an under-approximation of the logical state space: (i) a *folding* abstraction that collapses patterns into more general ones, (ii) an easy-to-check method to define (bisimilar) *equational abstractions*, and (iii) an *iterated bounded model checking* method that can detect if a logical state space within a given bound is *complete*. We also show that folding abstractions can be *faithful* for safety LTL properties, so that they do *not* generate any spurious counterexamples. These abstraction methods can be used in combination and, as we illustrate with examples, can be effective in making the logical state space finite. We have implemented these techniques in the Maude system, providing the first narrowing-based LTL model checker we are aware of.

1998 ACM Subject Classification D.2.4 Model Checking

Keywords and phrases model checking, infinite states, rewrite theories, narrowing

Digital Object Identifier 10.4230/LIPIcs.RTA.2013.81

1 Introduction

Model checking of finite-state systems is very well-developed (see. e.g., [2, 8]) and is well-supported by many tools and algorithms. Although model checking of infinite-state systems is more challenging, important advances have been made by various approaches, e.g. [1, 4, 11, 19, 22]. What many of these approaches have in common is the use of *symbolic* representations —such as formulas in a decidable logic or regular (string or tree) languages— to represent not just states but possibly infinite *sets of states*.

An intriguing possibility —first proposed in [31] for the simpler case of reachability analysis, and extended in [16] to LTL model checking— is to use rewriting-based symbolic techniques for infinite-state model checking by: (i) formalizing a concurrent system as a rewrite theory $\mathcal{R} = (\Sigma, E, R)$, whose states are elements of the initial algebra $\mathcal{T}_{\Sigma/E}$, and whose concurrent transitions are axiomatized by the rules R; (ii) representing possibly infinite sets of states by Σ-terms $t(x_1, \ldots, x_n)$ with *logical variables* $x_1, \ldots, x_n$, so that $t(x_1, \ldots, x_n)$ describes the set of its ground instances modulo E; and (iii) assuming an E-unification algorithm is available, exploring the *logical state space*, whose states are terms with logical variables $t(x_1, \ldots, x_n)$, by performing *narrowing with R modulo E* (see Section 2).

However, at the time the papers [16, 31] were published, no implementation of such narrowing-based logical model checking existed, and important open problems had to be resolved before its practical effectiveness could be demonstrated. This paper is all about solving such open problems by developing new narrowing-based model checking methods, and incorporating these new methods in an actual tool that can demonstrate the practical effectiveness of the narrowing-based model checking approach.

Open Problems Addressed. The main problems left unresolved by [16, 31] include:

1. *Dealing with infinite logical state spaces.* Narrowing can in general generate an infinite number of symbolic states; even though the idea of *folding* logical states by means of the subsumption $\preccurlyeq_E$ modulo E proposed in [16] showed that an infinite logical state space could sometimes be folded into a finite one, this was just one method, and no model checking algorithm existed when it failed.
2. *Dealing with a broad class of theories for which finitary unification algorithms exist.* The key point is that the equations E in a rewrite theory $\mathcal{R}$ must define not just structural axioms of the state, such as the associative-commutative nature of a set of processes, for which well-known unification algorithms exist: E must also define the truth values of the *state predicates* on which the temporal logic formulas are based. There is no hope that a finite set of unification algorithms can handle equations E of this kind: generic methods that can support a broad class of user-defined equational theories E are needed.
3. *Dealing with spurious counterexamples.* The use of abstractions typically brings with it spurious counterexamples that violate the given LTL formula on the abstract system but not in the concrete one. Can this spuriousness be avoided?

Our Contributions. Problem (1) is addressed in a twofold way by: (i) developing new abstraction techniques for logical state spaces that can be seamlessly combined with folding, such as *equational abstractions* (extended and simplified from their use in concrete state spaces in [30]) and the new *bisimilar equational abstractions* (Section 3.3); and (ii) developing a new *bounded* LTL *logical model checking algorithm* that does not require the state space to be finite, can model check a system up to a given depth, and can detect that a finite state space exists within the depth and support full verification in that case (Section 4).

Problem (2) is addressed by supporting equational theories E having the *finite variant property* [10] using the generic variant narrowing and unification algorithms in [17]; our approach is similar to that of the Maude-NPA [15], but is applied here to general LTL model checking, whereas Maude-NPA only supports reachability analysis for a restricted domain.

Problem (3) is addressed by showing that folding the logical state space by means of the subsumption $\preccurlyeq_E$ modulo E can give a *faithful* abstraction that does *not* generate any spurious counterexamples when verifying *safety LTL* properties (Section 3.2). This faithfulness of course holds when folding with $\preccurlyeq_E$ and bisimilar equational abstractions are used in combination. Note that folding abstractions are *strictly* more general than bisimulations since they are *not* faithful for general LTL properties.

Another important contribution is that all these new methods are supported by the new Maude LTL logical model checker that uses the Maude infrastructure [9, 12] for variant narrowing/unification and has many of its features implemented at the C++ level for efficiency reasons. We illustrate both the effectiveness of the tool and the new methods presented here by means of two nontrivial infinite-state systems: Lamport's bakery algorithm and Dijkstra's mutual exclusion algorithm for an unbounded number of processes (Section 5).

Related Work. Logical model checking is complementary to other infinite-state model checking techniques that symbolically represent a system's state space, such as regular languages [1], string/multiset grammars [4, 34], tree automata [22, 32], constraint logic programming [11], Presbuger arithmetic [6], program specialization [20], etc. Similar to logical model checking, they are often combined with abstraction methods, e.g., [5, 7, 21].

Our abstract logical model checking differs from these approaches in several ways. First, we do not impose restrictions in the formalisms used for the verification of properties. Except for requiring that equations have the finite variant property, the only condition imposed on the rewrite theories is being *topmost*, which is easily satisfied by many systems, including concurrent object-oriented systems. Second, the combination of different abstraction techniques can give a *faithful* abstraction that has *no* spurious counterexamples.

Similar to folding abstractions, there exist many infinite-state model checking methods to exploit an order relation $\preccurlyeq$, e.g., [14, 19]. However, those methods typically assume that $\preccurlyeq$ is well quasi-ordered (which implies well-foundedness of $\preccurlyeq$), while we do not impose such conditions on $\preccurlyeq$. Indeed, the E-subsumption relation $\preccurlyeq_E$ is, in general, *not* well-founded.

Bisimilar equational abstractions are also related to other abstraction techniques, e.g., [8, 25]. For rewrite theories, it is related to, and complements, abstraction techniques for rewrite theories such as [18, 30]. The main difference is that usual abstraction techniques do not provide bisimulations between the abstract and concrete systems and when they do provide them, they rely on manual proofs, instead than on simple, checkable criteria (such as those in Theorem 17) for defining bisimilar equational abstractions.

2 Preliminaries on Narrowing-based Logical Model Checking

An order-sorted signature is a triple $\Sigma = (S, \leq, \Sigma)$ with poset of sorts $(S, \leq)$ and operators Σ typed in $(S, \leq)$. The set $\mathcal{T}_\Sigma(\mathcal{X})_{\mathsf{s}}$ denotes the set of Σ-terms of sort s, and $\mathcal{T}_{\Sigma,\mathsf{s}}$ denotes the set of ground Σ-terms of sort s. *Positions* in a term t are denoted as strings of nonzero natural numbers that represent tree positions when t is parsed as a tree. A subterm of a term t at a position p is denoted by $t|_p$, and the replacement in t of such a subterm by another term u is denoted by $t[u]_p$. A *substitution* $\sigma : Y \to \mathcal{T}_\Sigma(\mathcal{X})$ is a function from $Y \subseteq \mathcal{X}$ to $\mathcal{T}_\Sigma(\mathcal{X})$ such that σy has the same sort as that of $y \in Y$. The substitution instance σt is a term obtained from t by *simultaneously* replacing each occurrence of variable $y \in Dom(\sigma)$ in t with σy.

A Σ-*equation* is an unoriented pair $t = t'$, where $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})_{\mathsf{s}}$ for some sort $s \in \Sigma$. Given a set E of Σ-equations, equational logic induces a congruence relation $=_E$ on terms $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$ [29]. The E-*subsumption* preorder $t \preccurlyeq_E t'$ holds iff there exists a substitution $\sigma : Y \to \mathcal{T}_\Sigma(\mathcal{X})$ such that $t =_E \sigma t'$, meaning that t' is *more general* than t modulo E. The E-renaming equivalence $t \approx_E t'$ holds iff there exists a substitution $\theta : \mathcal{X} \to \mathcal{X}$ such that $t =_E \theta t'$ and $\theta(x) \neq \theta(y)$ for any $x, y \in \mathcal{X}$, implying that $t \preccurlyeq_E t'$ and $t' \preccurlyeq_E t$.

An order-sorted *rewrite theory* is a triple $\mathcal{R} = (\Sigma, E, R)$ with Σ an order-sorted signature, E a set of Σ-equations, and R a set of rewrite rules, written $l \longrightarrow r$, where l, r are Σ-terms. Each rule $l \longrightarrow r$ specifies a *one-step rewrite* $t \longrightarrow_{R,E} t'$ iff there is a non-variable position p in t and a substitution σ such that $t|_p =_E \sigma l$ and $t' = t[\sigma r]_p$. A rewrite theory $\mathcal{R}$ specifies a concurrent system whose states are axiomatized 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), and whose concurrent transitions are axiomatized as one-step rewrites $\longrightarrow_{R,E}$ [28]. A rewrite theory $\mathcal{R}$ is *topmost* iff for each $l \longrightarrow r \in R$, $l, r \in \mathcal{T}_\Sigma(\mathcal{X})_{\mathsf{State}}$ for a sort State at the top of one of the connected component of $(S, \leq)$, and 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.

For a rewrite theory $\mathcal{R} = (\Sigma, E, R)$, state propositions can be defined by means of its equations. Each state proposition is defined as a term of sort Prop using (possibly parametric) function symbols of the form $p : \mathsf{s}_1 \ldots \mathsf{s}_n \to \mathsf{Prop}$, and the satisfaction relation is defined by equations using the auxiliary operator $\_\models\_ : \mathsf{State}\ \mathsf{Prop} \to \mathsf{Bool}$, where sort Bool has two constants *true* and *false* such that $\mathit{true} \neq_E \mathit{false}$ and for any term $t \in \mathcal{T}_{\Sigma,\mathsf{Bool}}$ of sort Bool, either $t =_E \mathit{true}$ or $t =_E \mathit{false}$ holds. By definition, a state proposition $p(u_1, \ldots, u_n) \in \mathcal{T}_{\Sigma/E,\mathsf{Prop}}$ is satisfied on $[t]_E \in \mathcal{T}_{\Sigma/E,\mathsf{State}}$ iff $(t \models p(u_1, \ldots, u_n)) =_E \mathit{true}$.

If $\mathcal{R}$ includes a set AP of state propositions *whose values on states are fully defined by its equations E*, we can associate to $\mathcal{R}$ a corresponding Kripke structure $\mathcal{K}(\mathcal{R})_{\mathit{AP}}$ for LTL model checking. A *Kripke structure* is a 4-tuple $\mathcal{K} = (S, \mathit{AP}, \mathcal{L}, \longrightarrow \mathcal{K})$ with S a set of *states*, AP a set of *atomic state propositions*, $\mathcal{L} : S \to \mathcal{P}(\mathit{AP})$ a *state-labeling function*, and $\longrightarrow_{\mathcal{K}} \subseteq S \times S$ a *total transition relation* where every state $s \in S$ has a next state $s' \in S$ with $s \longrightarrow_{\mathcal{K}} s'$. Given a subset $S_0 \subseteq S$, the set of its *successors* is $\mathit{Post}_{\mathcal{K}}(S_0) = \{s \in S \mid (\exists s_0 \in S_0)\ s_0 \longrightarrow_{\mathcal{K}} s\}$, and the set of its *reachable states* is $\mathit{Post}^*_{\mathcal{K}}(S_0) = \bigcup_{i \in \mathbb{N}} (\mathit{Post}_{\mathcal{K}})^i(S_0)$. We assume $\mathcal{R}$ is deadlock-free, since $\mathcal{R}$ can be transformed into an equivalent deadlock-free theory [30].

▶ **Definition 1.** Given $\mathcal{R} = (\Sigma, E, R)$ and a set AP of state propositions defined by its equations E, the corresponding Kripke structure is $\mathcal{K}(\mathcal{R})_{\mathit{AP}} = (\mathcal{T}_{\Sigma/E,\mathsf{State}}, \mathit{AP}, \mathcal{L}, \longrightarrow_{R,E})$, where $\mathcal{L}([t]_E) = \{p \in \mathit{AP} \mid (t \models p) =_E \mathit{true}\}$.

We present a topmost rewrite theory $\mathcal{R}$ specifying Lamport's bakery algorithm for mutual exclusion. Each state has the form "i ; j ; $[m_1] \ldots [m_n]$," where i is the current number in the bakery's number dispenser, j is the number currently being served, and the $[m_1] \ldots [m_n]$ are a multiset of customer processes, each in a *mode* m_l, which can be either *idle* (has not yet picked a number), or $\mathit{wait}(n)$ (waiting with number n), or $\mathit{crit}(n)$ (being served with number n). We model natural numbers as the free commutative monoid generated by 1 (denoted s) with multiset union (addition), denoted __ (empty syntax), satisfying *associativity*, *commutativity*, and *identity* ($\mathtt{0}$) axioms. For example, $0 = \mathtt{0}$, and $3 = s\,s\,s$. The behavior of the bakery algorithm is then specified by the following *topmost* rewrite rules:

```
rl [wake]: N ; M ; [idle] PS     =>  (s N) ; M ; [wait(N)] PS .
rl [crit]: N ; M ; [wait(M)] PS  =>  N ; M ; [crit(M)] PS .
rl [exit]: N ; M ; [crit(M)] PS  =>  N ; (s M) ; [idle] PS .
```

The state proposition `ex?` for the mutual exclusion is defined by the following equations, where the variable `WS` stands for a set of processes whose status is either *idle* or $\mathit{wait}(n)$:

```
eq N ; M ; WS |= ex? = true .
eq N ; M ; [crit(M1)] WS |= ex? = true .
eq N ; M ; [crit(M1)] [crit(M2)] PS |= ex? = false .
```

This system is infinite-state *in two ways*: (i) the counters i and j are unbounded; and (ii) the number n of customer processes is also unbounded. For example, given the initial state "$\mathtt{0}$; $\mathtt{0}$; [*idle*]," we obtain the infinite transition system of Figure 1.

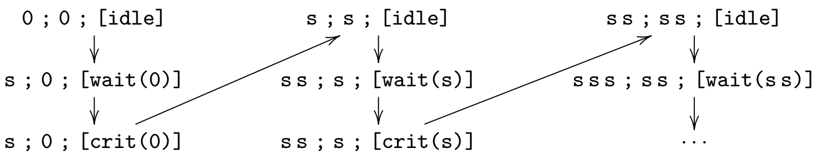

Figure 1 An infinite transition system for the Bakery algorithm from "`0 ; 0 ; [idle]`."

Narrowing [23, 24] generalizes term rewriting by allowing free variables in terms and by performing unification instead of matching. An *E-unifier* for an equation $t = t'$ is a substitution σ such that $\sigma t =_E \sigma t'$, and a set $CSU_E(t = t')$ of E-unifiers is *complete* iff any E-unifier ρ for $t = t'$ has a more general substitution σ in $CSU_E(t = t')$, i.e., there is a substitution η such that $\rho =_E \sigma \circ \eta$. Given a *topmost* rewrite theory $\mathcal{R} = (\Sigma, E, R)$, if the left-hand sides of the rules R are non-variable terms and a finitary E-unification procedure is available, each rule $l \longrightarrow r$ specifies a *topmost narrowing step* $t \rightsquigarrow_{\sigma,R,E} t'$ (or $t \rightsquigarrow_{R,E} t'$) iff there exists an E-unifier $\sigma \in CSU_E(t = l)$ such that $t' = \sigma r$.

If an equational theory (Σ, E) has the *finite variant property*, there is an algorithm to compute a finitary and complete set $CSU_E(t = t')$ of E-unifiers [17]. An E-variant of a term t is a pair (t', θ) with t' an E-canonical form of a substitution instance θt, i.e., $\theta t \longrightarrow_E^* t'$ and t' cannot be further rewritten. A variant (t_2, θ_2) is more general than (t_1, θ_1) iff there is a substitution η such that $t_1 =_E \eta t_2$ and $\theta_1 =_E \theta_2 \circ \eta$. An equational theory (Σ, E) has the finite variant property iff the set of most general E-variants for each term is finite (see [10, 17] for details). For the Bakery example, the equations for the state proposition `ex?` trivially satisfy the finite variant property because their right-hand sides are all constants.

Such a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ also specifies a *logical* transition system $\mathcal{N}_{\mathcal{R}}$ [16] whose states are elements of the algebra $\mathcal{T}_{\Sigma/E}(\mathcal{X})$ of sort State (excluding variables as states), and whose transitions are specified by topmost narrowing steps $\rightsquigarrow_{R,E}$. That is, the states of $\mathcal{N}_{\mathcal{R}}$ are not *concrete states* (i.e., ground terms), but *state patterns*, that is, terms $t(x_1, \ldots, x_n)$ with *logical variables* $x_1, \ldots, x_n$. What $t(x_1, \ldots, x_n)$ stands for is *not* a single state, but the set of all concrete states $[\theta t] \in \mathcal{T}_{\Sigma/E,\mathsf{State}}$ that are its *ground instances*.

If a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ defines a *finite* set *AP* of state propositions by its equations E, we can also associate to $\mathcal{R}$ a corresponding *narrowing-based logical Kripke structure* $\mathcal{N}_{\mathcal{R}}^{AP}$. Each state in the underlying logical transition system $\mathcal{N}_{\mathcal{R}}$ is now split into possibly several states in $\mathcal{N}_{\mathcal{R}}^{AP}$ (by $\rightsquigarrow_{AP}$ in the following definition) so that the truth of every state proposition for each state in $\mathcal{N}_{\mathcal{R}}^{AP}$ is decided into *true* or *false*.

▶ **Definition 2.** [16] Given a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ and a finite set $AP = \{p_1, \ldots, p_n\}$ of state propositions defined by E, its narrowing-based *logical Kripke structure* is:

$$\mathcal{N}_{\mathcal{R}}^{AP} = \big(N_{\mathcal{R}}^{AP},\ AP,\ \mathcal{L},\ (\rightsquigarrow_{R,E};\rightsquigarrow_{AP})\big)$$

where $t\ (\rightsquigarrow_{R,E};\rightsquigarrow_{AP})\ t' \iff (\exists u)\ t \rightsquigarrow_{R,E} u \rightsquigarrow_{AP} t'$, and

- $N_{\mathcal{R}}^{AP} = \{[t]_E \in \mathcal{T}_{\Sigma/E}(\mathcal{X})_{\mathsf{State}} - \mathcal{X} \mid (\forall p \in AP)\ (t \models p) =_E \mathit{true} \vee (t \models p) =_E \mathit{false}\}$,
- $\mathcal{L}([t]_E) = \{p \in AP \mid (t \models p) =_E \mathit{true}\}$,
- $t \rightsquigarrow_{AP} t' \iff \exists\theta \in CSU_E\big((t \models p_1) = w_1 \wedge \cdots \wedge (t \models p_n) = w_n\big)$ such that $t' = \theta t$, where for each $1 \leq i \leq n$, w_i is either *true* or *false*.

For the Bakery example, given the *logical* initial state `N ; N ; [idle]`, we obtain within $\mathcal{N}_{\mathcal{R}}$ the infinite transition system in Figure 2. The corresponding Kripke structure $\mathcal{N}_{\mathcal{R}}^{\{\texttt{ex?}\}}$ is then similar to $\mathcal{N}_{\mathcal{R}}$, but each logical state in $\mathcal{N}_{\mathcal{R}}$ is split according to the truth of `ex?`.

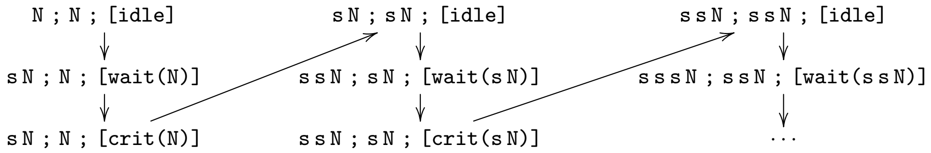

Figure 2 An infinite transition system with *logical* states for the Bakery algorithm.

Such a narrowing-based Kripke structure $\mathcal{N}^{AP}_{\mathcal{R}}$ can be considered as an *exact abstraction* of the concrete Kripke structure $\mathcal{K}(\mathcal{R})_{AP}$, where concrete states are abstracted by means of the *E-subsumption* preorder $\preccurlyeq_E$ as shown in the following theorem.

▶ **Theorem 3.** *[16] Given a topmost rewrite theory $\mathcal{R} = (\Sigma, E, R)$ and a finite set AP of state propositions defined by E, for an LTL formula φ and a pattern $t \in N^{AP}_{\mathcal{R}}$:*

$$\mathcal{N}^{AP}_{\mathcal{R}}, [t]_E \models \varphi \iff (\forall \theta : \mathcal{X} \to \mathcal{T}_\Sigma)\ \mathcal{K}(\mathcal{R})_{AP}, [\theta t]_E \models \varphi.$$

However, $\mathcal{N}^{AP}_{\mathcal{R}}$ often has an infinite number of *logical* states as in Figure 2. The following sections explain how we can reduce it to a finite state space by abstraction techniques that provide either an over-approximation or an under-approximation of $\mathcal{N}^{AP}_{\mathcal{R}}$.

3 Abstract Logical Model Checking

For model checking techniques, an abstraction $\widehat{\mathcal{K}}$ of a concurrent system typically preserves all moves of the original system $\mathcal{K}$, in terms of a *simulation* between $\mathcal{K}$ and $\widehat{\mathcal{K}}$. Given two Kripke structures $\mathcal{K}_i = (S_i, AP, \mathcal{L}_i, \longrightarrow_{\mathcal{K}_i})$, $i = 1, 2$, a binary relation $H \subseteq S_1 \times S_2$ is a *simulation* iff (i) $s_1 H s_2$ and $s_1 \longrightarrow_{\mathcal{K}_1} s_1'$ implies that $(\exists s_2' \in S_2)\ s_1' H s_2'$ and $s_2 \longrightarrow_{\mathcal{K}_2} s_2'$, and (ii) $s_1 H s_2 \implies \mathcal{L}_1(s_1) = \mathcal{L}_2(s_2)$. A simulation $H \subseteq S_1 \times S_2$ is *total* iff for any $s_1 \in S_1$ there exists $s_2 \in S_2$ such that $s_1 H s_2$. H is a *bisimulation* iff both H and H^{-1} are simulations. If $\mathcal{K}_2$ *simulates* $\mathcal{K}_1$, any LTL formula satisfied in $\mathcal{K}_2$ is also satisfied in $\mathcal{K}_1$.

▶ **Lemma 4.** *[8] Given $\mathcal{K}_i = (S_i, AP, \mathcal{L}_i, \longrightarrow_{\mathcal{K}_i})$, $i = 1, 2$, for a simulation $H \subseteq S_1 \times S_2$, if $s^1_0 H s^2_0$, then for any LTL formula φ over AP, $\mathcal{K}_2, s^2_0 \models \varphi$ implies $\mathcal{K}_1, s^1_0 \models \varphi$.*

This section presents two abstraction techniques for narrowing-based logical model checking, namely, *folding abstractions* and *equational abstractions*. Such an abstraction $\widehat{\mathcal{K}}$ provides an over-approximation of the original system $\mathcal{K}$, i.e., $\widehat{\mathcal{K}}$ *simulates* $\mathcal{K}$. Thus, if we verify that φ holds for $\widehat{\mathcal{K}}$, then we can be sure that it also holds for $\mathcal{K}$. However, as usual for over-approximation abstraction techniques, a counterexample in $\widehat{\mathcal{K}}$ can be *spurious*, so that it has no counterpart in $\mathcal{K}$. We also provide some conditions for both abstraction techniques when $\widehat{\mathcal{K}}$ can be *faithful* for a certain *subset* Δ of LTL formulas so that $\widehat{\mathcal{K}}$ generates *no* spurious counterexamples, i.e., for each $\varphi \in \Delta$ and $s\, H\, \hat{s}$, $\widehat{\mathcal{K}}, \hat{s} \models \varphi$ iff $\mathcal{K}, s \models \varphi$.

3.1 Folding Abstractions

We can reduce a logical Kripke structure by collapsing each state into a more *general* state according to the E-subsumption preorder $\preccurlyeq_E$, by the notion of folding abstraction proposed in [16]. In this paper we further generalize folding abstractions with any *folding preorder* $\preccurlyeq$.

▶ **Definition 5.** Given a Kripke structure $\mathcal{K} = (S, AP, \mathcal{L}, \longrightarrow_{\mathcal{K}})$, a *folding preorder* $\preccurlyeq\ \subseteq S^2$ is a reflexive and transitive relation on S that defines a simulation between $\mathcal{K}$ and $\mathcal{K}$.

For a narrowing-based Kripke structure $\mathcal{N}^{AP}_{\mathcal{R}}$, the most common folding relations are: equality modulo E, renaming modulo E, and matching modulo E, i.e., $\preccurlyeq\ \in \{=_E, \approx_E, \preccurlyeq_E\}$ [16].

We can iteratively construct a *folding abstraction* of a Kripke structure $\mathcal{K}$ from a set of initial states $I \subseteq S$, using a folding preorder $\preccurlyeq\ \subseteq S^2$ as shown in Definition 6 below. Each state $s \in S$ in $\mathcal{K}$ is collapsed into a *previously seen* state $t \in S$ such that $s \preccurlyeq t$, while any transition for the folded state s is transferred to the state t in the folding abstraction. Such a folded Kripke structure has in general much fewer states than the original structure, and can sometimes collapse an infinite-state space to a finite-state one.

▶ **Definition 6** (Folding Abstraction). Given $\mathcal{K} = (S, AP, \mathcal{L}, \longrightarrow_{\mathcal{K}})$, a folding preorder $\preccurlyeq \subseteq S^2$, and a set of initial states $I \subseteq S$, the *folding abstraction* of $\mathcal{K}$ from I is the Kripke structure

$$\mathcal{R}each^{\preccurlyeq}_{\mathcal{K}}(I) = (Post^*_{\mathcal{K}\preccurlyeq}(I),\ AP,\ \mathcal{L},\ \longrightarrow_{\mathcal{R}each^{\preccurlyeq}_{\mathcal{K}}(I)})$$

where $Post^*_{\mathcal{K}\preccurlyeq}(I) = \bigcup_{i\in\mathbb{N}} Post^i_{\mathcal{K}\preccurlyeq}(I)$ and $\longrightarrow_{\mathcal{R}each^{\preccurlyeq}_{\mathcal{K}}(I)} = \bigcup_{i\in\mathbb{N}} \longrightarrow^{\preccurlyeq}_{\mathcal{K},i}$ such that:

- $Post^{n+1}_{\mathcal{K}\preccurlyeq}(I)$ is the successor set of $Post^n_{\mathcal{K}\preccurlyeq}(I)$ *not* subsumed by previously seen states:

$$Post^0_{\mathcal{K}\preccurlyeq}(I) = I, \quad Post^{n+1}_{\mathcal{K}\preccurlyeq}(I) = \{s \in Post_{\mathcal{K}}(Post^n_{\mathcal{K}\preccurlyeq}(I)) \mid \forall l \leq n\ \forall u \in Post^l_{\mathcal{K}\preccurlyeq}(I).\ s \not\preccurlyeq u\}.$$

- $\longrightarrow^{\preccurlyeq}_{\mathcal{K},n+1} \subseteq Post^n_{\mathcal{K}\preccurlyeq}(I) \times \left[\bigcup_{0\leq i\leq n+1} Post^i_{\mathcal{K}\preccurlyeq}(I)\right]$ defines the transitions from each state $s \in Post^n_{\mathcal{K}\preccurlyeq}(I)$ to a next state $t \in Post^l_{\mathcal{K}\preccurlyeq}(I)$ for $0 \leq l \leq n+1$, up to $n+1$ steps:

$$\longrightarrow^{\preccurlyeq}_{\mathcal{K},0} = \emptyset, \qquad s \longrightarrow^{\preccurlyeq}_{\mathcal{K},n+1} s' \iff \exists t \in Post_{\mathcal{K}}(s).\ t \preccurlyeq s'.$$

Each reachable state $s \in Post^*_{\mathcal{K}}(I)$ in a Kripke structure $\mathcal{K}$ has a corresponding abstract state $\hat{s} \in Post^*_{\mathcal{K}\preccurlyeq}(I)$ in the folding abstraction $\mathcal{R}each^{\preccurlyeq}_{\mathcal{K}}(I)$ as follows.

▶ **Lemma 7.** *Given $\mathcal{K} = (S, AP, \mathcal{L}, \longrightarrow \mathcal{K})$, a folding preorder $\preccurlyeq \subseteq S^2$, and a set of initial states $I \subseteq S$, for each reachable state $s \in Post^*_{\mathcal{K}}(I)$, there is $\hat{s} \in Post^*_{\mathcal{K}\preccurlyeq}(I)$ such that $s \preccurlyeq \hat{s}$.*

Proof. For a reachable state $s \in Post^*_{\mathcal{K}}(I)$ of $\mathcal{K}$, there exists a finite path $\pi_s : [n] \to S$ with length $n \in \mathbb{N}$ beginning in I and ending at s (i.e., $\pi_s(0) \in I$ and $\pi_s(n-1) = s$), where $[n] = \{0, 1, \ldots, n\}$. We show this lemma by induction on the length of π_s. First, if $|\pi_s| = 0$, then $s \in I = Post^0_{\mathcal{K}\preccurlyeq}(I) \subseteq Post^*_{\mathcal{K}\preccurlyeq}(I)$, and $s \preccurlyeq s$. Next, suppose that for any path π beginning in I with length n, there exists an abstract state $t_{n-1} \in Post^*_{\mathcal{K}\preccurlyeq}(I)$ such that $\pi(n-1) \preccurlyeq t_{n-1}$. Consider a path π_s with length $n+1$ such that $\pi_s(0) \in I$ and $\pi_s(n) = s$. By induction hypothesis, there exists $t_{n-1} \in Post^*_{\mathcal{K}\preccurlyeq}(I)$ such that $\pi_s(n-1) \preccurlyeq t_{n-1}$. Notice that $t_{n-1} \in Post^k_{\mathcal{K}\preccurlyeq}(I)$ for some $k \in \mathbb{N}$. Since $\preccurlyeq$ is a simulation between $\mathcal{K}$ and $\mathcal{K}$:

$$\begin{array}{ccccccc}
\pi_s(n-1) & \in & Post^*_{\mathcal{K}}(I) & \longrightarrow_{\mathcal{K}} & \pi_s(n) & \in & Post^*_{\mathcal{K}}(I) \\
\curlyeqprec & & & & \curlyeqprec & & \\
t_{n-1} & \in & Post^k_{\mathcal{K}\preccurlyeq}(I) & \longrightarrow_{\mathcal{K}} & \exists\, t_n & \in & Post_{\mathcal{K}}(Post^k_{\mathcal{K}\preccurlyeq}(I))
\end{array}$$

There are now two possibilities: (i) if $t_n \in Post^{k+1}_{\mathcal{K}\preccurlyeq}(I)$, we found $t_n \in Post^*_{\mathcal{K}\preccurlyeq}(I)$ such that $s = \pi_s(n) \preccurlyeq t_n$; (ii) otherwise, there exist $l \leq k$ and $u \in Post^l_{\mathcal{K}\preccurlyeq}(I)$ such that $t_n \preccurlyeq u$, since by definition $Post^{k+1}_{\mathcal{K}\preccurlyeq}(I) = \{s \in Post_{\mathcal{K}}(Post^k_{\mathcal{K}\preccurlyeq}(I)) \mid \forall l \leq k\ \forall u \in Post^l_{\mathcal{K}\preccurlyeq}(I).\ s \not\preccurlyeq u\}$; that is, we found $u \in Post^*_{\mathcal{K}\preccurlyeq}(I)$ such that $s = \pi_s(n) \preccurlyeq t_n \preccurlyeq u$. ◀

Furthermore, the folding abstraction $\mathcal{R}each^{\preccurlyeq}_{\mathcal{K}}(I)$ of $\mathcal{K} = (S, AP, \mathcal{L}, \longrightarrow_{\mathcal{K}})$ simulates the *reachable* substructure $\mathcal{R}each_{\mathcal{K}}(I)$ of $\mathcal{K}$ that only contains reachable states from I, where $\mathcal{R}each_{\mathcal{K}}(I) = (Post^*_{\mathcal{K}}(I),\ AP,\ \mathcal{L},\ \longrightarrow_{\mathcal{K}} \cap\, (Post^*_{\mathcal{K}}(I))^2)$.

▶ **Theorem 8.** *Given $\mathcal{K} = (S, AP, \mathcal{L}, \longrightarrow \mathcal{K})$, a folding preorder $\preccurlyeq \subseteq S^2$, and a set of initial states $I \subseteq S$, the folding preorder $\preccurlyeq$ is a total simulation between $\mathcal{R}each_{\mathcal{K}}(I)$ and $\mathcal{R}each^{\preccurlyeq}_{\mathcal{K}}(I)$.*

Proof. Suppose $s \preccurlyeq t$ and $s \longrightarrow_{\mathcal{K}} s'$ for states $s, s' \in Post^*_{\mathcal{K}}(I)$ and an abstract state $t \in Post^*_{\mathcal{K}\preccurlyeq}(I)$. By definition, $t \in Post^k_{\mathcal{K}\preccurlyeq}(I)$ for some $k \in \mathbb{N}$. Since $\preccurlyeq$ is a simulation between $\mathcal{K}$ and $\mathcal{K}$, there exists $t' \in Post_{\mathcal{K}}(Post^k_{\mathcal{K}\preccurlyeq}(I))$ such that $t \longrightarrow_{\mathcal{K}} t'$ and $s' \preccurlyeq t'$. There are also two possibilities: (i) if $t' \in Post^{k+1}_{\mathcal{K}\preccurlyeq}(I)$, we have $t \longrightarrow^{\preccurlyeq}_{\mathcal{K},k+1} t'$ such that $s' \preccurlyeq t'$; (ii) otherwise, by definition of $Post^{k+1}_{\mathcal{K}\preccurlyeq}(I)$, there exist $l \leq k$ and $t'' \in Post^l_{\mathcal{K}\preccurlyeq}(I)$ such that $t' \preccurlyeq t''$, and we have $t \longrightarrow^{\preccurlyeq}_{\mathcal{K},k+1} t''$ again, where $s' \preccurlyeq t' \preccurlyeq t''$. Therefore, $\preccurlyeq$ is a simulation between $\mathcal{R}each_{\mathcal{K}}(I)$ and $\mathcal{R}each^{\preccurlyeq}_{\mathcal{K}}(I)$. Also, $\preccurlyeq$ is total by Lemma 7. ◀

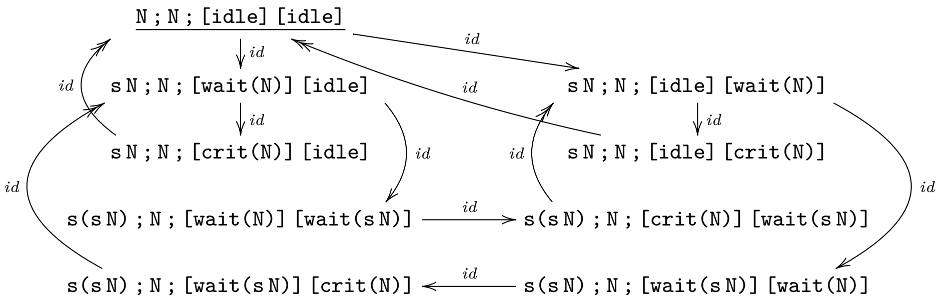
N ; N ; [idle] [idle]
id
id
sN ; N ; [wait(N)] [idle]
id
sN ; N ; [idle] [wait(N)]
id
sN ; N ; [crit(N)] [idle]
id
id
sN ; N ; [idle] [crit(N)]
id
id
s(sN) ; N ; [wait(N)] [wait(sN)]
id
s(sN) ; N ; [crit(N)] [wait(sN)]
id
s(sN) ; N ; [wait(sN)] [crit(N)]
id
s(sN) ; N ; [wait(sN)] [wait(N)]
id
id

Figure 3 A finite folding abstraction with a folding preorder $\preccurlyeq_E$ for the Bakery algorithm. We add a double-headed arrow between states A and C to denote both a transition from state A to another state B and the fact that state B is folded into state C.

For our Bakery example, given the *logical* initial state N ; N ; [*idle*] [*idle*], Figure 3 shows the *finite* folding abstraction of $\mathcal{N}_{\mathcal{R}}^{\{\texttt{ex?}\}}$ with the folding preorder $\preccurlyeq_E$. The mutual exclusion $\Box$ex? is satisfied in the folding abstraction, since the state proposition ex? evaluates to *true* in every state. Thanks to Theorem 8, $\Box$ex? is satisfied for any possible instance of it.

3.2 Faithfulness of Folding Abstractions

A folding abstraction $\mathcal{R}each_{\mathcal{K}}^{\preccurlyeq}(I)$ is in general an over-approximation of a logical state space. If an LTL formula φ is *not* satisfied in $\mathcal{R}each_{\mathcal{K}}^{\preccurlyeq}(I)$, it can generate a spurious counterexample for φ. Nonetheless, if a folding preorder $\preccurlyeq$ is *symmetric*, then $\preccurlyeq$ becomes a total bisimulation by Theorem 8, so that both satisfy exactly the same set of LTL formulas. For example, both $=_E$ and $\approx_E$ are symmetric for a narrowing-based Kripke structure.

What can we then say about $\preccurlyeq_E$ for a narrowing-based Kripke structure? Since $\preccurlyeq_E$ is more general than $=_E$ and $\approx_E$, it has a better chance to yield a finite state space, although it may generate a spurious counterexample for an LTL formula. However, a folding abstraction is *faithful* for invariants; that is, if there is a counterexample for any invariant $\Box\Phi$ in $\mathcal{R}each_{\mathcal{K}}^{\preccurlyeq}(I)$, where Φ is a boolean formula with no temporal operators, there exists a *real* counterexample in $\mathcal{K}$. This faithfulness follows from the fact that each state in $\mathcal{R}each_{\mathcal{K}}^{\preccurlyeq}(I)$ is still *reachable* from I in the original Kripke structure $\mathcal{K}$.

▶ **Lemma 9.** *Given a Kripke structure $\mathcal{K} = (S, AP, \mathcal{L}, \longrightarrow_{\mathcal{K}})$, a folding preorder $\preccurlyeq \subseteq S^2$, and a set of initial states $I \subseteq S$, we have $Post^*_{\mathcal{K}_\preccurlyeq}(I) \subseteq Post^*_{\mathcal{K}}(I)$.*

Proof. Recall that $Post^*_{\mathcal{K}_\preccurlyeq}(I) = \bigcup_{i\in\mathbb{N}} Post^i_{\mathcal{K}_\preccurlyeq}(I)$. By definition, $Post^0_{\mathcal{K}_\preccurlyeq}(I) = I \subseteq Post^*_{\mathcal{K}}(I)$. Suppose that $Post^n_{\mathcal{K}_\preccurlyeq}(I) \subseteq Post^*_{\mathcal{K}}(I)$ for some $n \in \mathbb{N}$. Since $Post^{n+1}_{\mathcal{K}_\preccurlyeq}(I) \subseteq Post_{\mathcal{K}}(Post^n_{\mathcal{K}_\preccurlyeq}(I))$, for each $s' \in Post^{n+1}_{\mathcal{K}_\preccurlyeq}(I)$, there exists $s \in Post^n_{\mathcal{K}_\preccurlyeq}(I)$ such that $s \longrightarrow_{\mathcal{K}} s'$. By induction hypothesis, $s \in Post^*_{\mathcal{K}}(I)$, and thus $s' \in Post^*_{\mathcal{K}}(I)$. Therefore, $Post^{n+1}_{\mathcal{K}_\preccurlyeq}(I) \subseteq Post^*_{\mathcal{K}}(I)$. ◀

If a folding abstraction $\mathcal{R}each_{\mathcal{K}}^{\preccurlyeq}(I)$ does *not* satisfy an invariant, then there exists an *error* state $s \in Post^*_{\mathcal{K}_\preccurlyeq}(I)$ in $\mathcal{R}each_{\mathcal{K}}^{\preccurlyeq}(I)$ that violates the invariant. Because the error state s is again reachable from I in the original Kripke structure $\mathcal{K}$, we can construct a *concrete counterexample* in $\mathcal{K}$ by backward search from s to I. Consequently:

▶ **Theorem 10** (Faithfulness for Invariants)**.** *Given a Kripke structure $\mathcal{K} = (S, AP, \mathcal{L}, \longrightarrow \mathcal{K})$, a folding preorder $\preccurlyeq \subseteq S^2$, and a set of initial states $I \subseteq S$, for any invariant $\Box\Phi$:*

$$\mathcal{R}each_{\mathcal{K}}^{\preccurlyeq}(I), I \models \Box\Phi \iff \mathcal{K}, I \models \Box\Phi.$$

Moreover, folding abstractions can provide a faithful model checking procedure for *safety* LTL formulas. For a safety LTL formula φ, there exists a *finite automaton* $\mathcal{F}_{\neg\varphi}$ that recognizes counterexamples for φ [2, 26]. A finite automaton is a 5-tuple $\mathcal{F} = (Q, Q_0, \mathcal{P}(AP), \delta, F)$ with Q a finite set of states, $Q_0 \subseteq Q$ a set of initial states, $\mathcal{P}(AP)$ an alphabet of transition labels, $\delta \subseteq Q \times \mathcal{P}(AP) \times Q$ a transition relation, and $F \subseteq Q$ a set of final states. The *language* accepted by $\mathcal{F}$ is the set $L(\mathcal{F})$ of finite runs of $\mathcal{F}$ starting in Q_0 and ending in F. Given a Kripke structure $\mathcal{K}$ and a set $I \subseteq S$ of initial states, the *synchronous product* of $\mathcal{K}$ and $\mathcal{F}$ is a finite automaton $\mathcal{K}[I] \times \mathcal{F} = (S \times Q, I \times Q_0, \mathcal{P}(AP), \delta_{\mathcal{K}}, S \times F)$ such that $(s, b) \xrightarrow{\mathcal{L}(s)} (s', b') \in \delta_{\mathcal{K}}$ iff $s \longrightarrow_{\mathcal{K}} s' \,\wedge\, b \xrightarrow{\mathcal{L}(s)} b' \in \delta$. The model checking problem of a *safety* LTL formula φ can then be characterized by using a finite automaton $\mathcal{F}_{\neg\varphi}$ associated to the negated formula $\neg\varphi$, where $\mathcal{K}, I \models \varphi$ iff $L(\mathcal{K}[I] \times \mathcal{F}_{\neg\varphi}) = \emptyset$ [2, 26].

Since the emptiness checking of the finite automaton $\mathcal{K}[I] \times \mathcal{F}_{\neg\varphi}$ can be characterized by the reachability analysis of the final states, we can apply our previous result to *faithfully* abstract the synchronous product $\mathcal{K}[I] \times \mathcal{F}_{\neg\varphi}$. For a folding preorder $\preccurlyeq$ of $\mathcal{K}$, let the *product preorder* $\preccurlyeq_{\mathcal{F}} \subseteq (S \times Q)^2$ be defined by the equivalence: $(s, b) \preccurlyeq_{\mathcal{F}} (s', b') \iff s \preccurlyeq s' \,\wedge\, b = b'$.

▶ **Lemma 11.** *Given a finite automaton $\mathcal{F}$ and a folding preorder $\preccurlyeq$ for a Kripke structure $\mathcal{K}$, the product preorder $\preccurlyeq_{\mathcal{F}}$ is a folding preorder for the synchronous product $\mathcal{K}[I] \times \mathcal{F}$.*

Proof. Suppose that $(s_1, b_1) \xrightarrow{\mathcal{L}(s_1)} (s_1', b_1') \in \delta_{\mathcal{K}}$ and $(s_1, b_1) \preccurlyeq_{\mathcal{F}} (s_2, b_2)$. By definition, $s_1 \preccurlyeq s_2$ and $b_1 = b_2$. Since $\preccurlyeq$ is a simulation between $\mathcal{K}$ and $\mathcal{K}$, there exists $s_2' \in S$ such that $s_2 \longrightarrow_{\mathcal{K}} s_2'$ and $s_1' \preccurlyeq s_2'$, and $\mathcal{L}(s_1) = \mathcal{L}(s_2)$. Hence, $(s_2, b_2) \xrightarrow{\mathcal{L}(s_1)} (s_2', b_1') \in \delta_{\mathcal{K}}$, and $(s_1', b_1') \preccurlyeq_{\mathcal{F}} (s_2', b_1')$. Therefore, $\preccurlyeq_{\mathcal{F}}$ is a simulation between $\mathcal{K}[I] \times \mathcal{F}$ and $\mathcal{K}[I] \times \mathcal{F}$. ◀

Therefore, by Theorem 10, for a safety LTL formula φ, $L\big(\mathcal{R}each^{\preccurlyeq_{\mathcal{F}_{\neg\varphi}}}_{\mathcal{K}[I] \times \mathcal{F}_{\neg\varphi}}(I \times Q_0)\big) = \emptyset$ iff $L(\mathcal{K}[I] \times \mathcal{F}_{\neg\varphi}) = \emptyset$. Consequently, we have:

▶ **Theorem 12** (Faithfulness for Safety Properties)**.** *Given $\mathcal{K} = (S, AP, \mathcal{L}, \longrightarrow_{\mathcal{K}})$, a folding preorder $\preccurlyeq \subseteq S^2$, and a set of initial states $I \subseteq S$, for a* safety *LTL formula φ, there exists a* finite *automaton $\mathcal{F}_{\neg\varphi}$ with Q_0 a set of initial states such that:*

$$L\big(\mathcal{R}each^{\preccurlyeq_{\mathcal{F}_{\neg\varphi}}}_{\mathcal{K}[I] \times \mathcal{F}_{\neg\varphi}}(I \times Q_0)\big) = \emptyset \iff \mathcal{K}, I \models \varphi.$$

3.3 Equational Abstractions for Logical State Space

A logical state representation for a rewrite theory $\mathcal{R}$ already affords a huge abstraction, and it may turn an infinite system into a more manageable system. However, even a folding abstraction of such a logical state space need not be finite in general. For example, when we consider our Bakery example and the logical initial state `N ; N ; IS` that does not bound the number of customer processes, the folding abstraction with $\preccurlyeq_E$ has an infinite path that keeps incrementing the number of processes with instantiation, as shown in Figure 4.

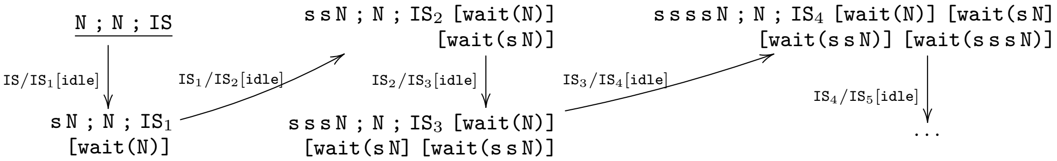
N ; N ; IS
IS/IS₁[idle]
s N ; N ; IS₁ [wait(N)]
IS₁/IS₂[idle]
s s N ; N ; IS₂ [wait(N)] [wait(s N)]
IS₂/IS₃[idle]
s s s N ; N ; IS₃ [wait(N)] [wait(s N)] [wait(s s N)]
IS₃/IS₄[idle]
s s s s N ; N ; IS₄ [wait(N)] [wait(s N)] [wait(s s N)] [wait(s s s N)]
IS₄/IS₅[idle]
. . .

Figure 4 An infinite folded logical transition system for the Bakery algorithm with an arbitrary number of processes. The logical variable $\mathtt{IS}_k$ stands for a set of `idle` processes.

An abstraction of a concurrent system can be constructed by a suitable equivalence relation $\equiv$ on states [8, 30]. Given $\mathcal{K} = (S, AP, \mathcal{L}, \longrightarrow_{\mathcal{K}})$ and an equivalence relation $\equiv \subseteq S \times S$ such that $s_1 \equiv s_2$ implies $\mathcal{L}(s_1) = \mathcal{L}(s_2)$, the *quotient abstraction* $\mathcal{K}/\!\equiv$ is a Kripke structure $(S/\!\equiv, AP, \mathcal{L}, \longrightarrow_{\mathcal{K}/\equiv})$ where $[s_1] \longrightarrow_{\mathcal{K}/\equiv} [s_2]$ iff $(\exists s_1' \in [s_1], s_2' \in [s_2])\ s_1' \longrightarrow_{\mathcal{K}} s_2'$. For a topmost and deadlock-free rewrite theory $\mathcal{R} = (\Sigma, E, R)$ containing a set AP of state propositions defined by the equations E, by adding a set of extra equations G to $\mathcal{R}$, we can define an *equational abstraction* $\mathcal{R}/G = (\Sigma, E \cup G, R)$ [30], which specifies the quotient abstraction of $\mathcal{K}(\mathcal{R})_{AP}$ by the equivalence relation $\equiv_G$ on states, namely, $[t]_E \equiv_G [t']_E \iff t =_{E\cup G} t'$, where $[t]_E \equiv_G [t']_E$ implies $\mathcal{L}([t]_E) = \mathcal{L}([t']_E)$.

In this section we explain how equational abstractions can be applied for narrowing-based model checking for collapsing an infinite *logical* state space into a finite one, whereas equation abstractions [30] are used for ground terms in the literature for non-logical state spaces.

▶ **Definition 13** (Equational Abstraction). Given $\mathcal{R} = (\Sigma, E, R)$ and a set of equations G, the rewrite theory $\mathcal{R}/G = (\Sigma, E \cup G, R)$ defines an *equational abstraction* iff: (i) finitary unification procedures modulo E and modulo $E \cup G$ are available, and (ii) $true \neq_{E\cup G} false$.

If a set AP of state propositions is fully defined by E, whenever $t =_{E\cup G} t'$ for two states $t, t' \in \mathcal{T}_{\Sigma/E,\mathsf{State}}$, condition (ii) ensures that both t and t' satisfy exactly the same state propositions. Note that if $E \cup G$ has the finite variant property, there is a finitary unification procedure module $E \cup G$ as explained in [10, 17], which is available in the Maude system [12].

Similar to equational abstractions for a concrete Kripke structure $\mathcal{K}(\mathcal{R})_{AP}$, we obtain a simulation between a logical Kripke structure $\mathcal{N}_{\mathcal{R}}^{AP}$ and its equational abstraction $\mathcal{N}_{\mathcal{R}/G}^{AP}$.

▶ **Lemma 14.** *Given a topmost rewrite theory $\mathcal{R} = (\Sigma, E, R)$, a finite set AP of state propositions defined by E, and a set G of equations, if $\mathcal{R}/G$ is an equational abstraction, then $H_G = \{([t]_E, [t]_{E\cup G}) \mid t \in \mathcal{T}_\Sigma(\mathcal{X})_{\mathsf{State}}\}$ is a simulation between $\mathcal{N}_{\mathcal{R}}^{AP}$ and $\mathcal{N}_{\mathcal{R}/G}^{AP}$.*

Proof. For $t \in \mathcal{T}_\Sigma(\mathcal{X})$ and $u \in \mathcal{T}_{\Sigma/G}(\mathcal{X})$, suppose $t =_{E\cup G} u$ and $t \rightsquigarrow_{\theta,R,E} t'$ using rule $l \longrightarrow r \in R$, that is, $\theta \in CSU_E(t = l)$ and $t' = \theta(r)$. Since $\theta \in CSU_E(t = l)$, there exists $\theta' \in CSU_{E\cup G}(u = l)$ such that $\theta =_{E\cup G} \theta'$. Therefore, for $u' = \theta' r$, $u \rightsquigarrow_{\theta',R,E\cup G} u'$ using the same rule $l \longrightarrow r \in R$ and $t' = \theta r =_{E\cup G} \theta' r = u'$. ◀

We introduce *bisimilar equational abstractions*, which ensure a bisimulation between the narrowing-based Kripke structure $\mathcal{N}_{\mathcal{R}}^{AP}$ and its quotient abstraction $\mathcal{N}_{\mathcal{R}/G}^{AP}$.

▶ **Definition 15** (Bisimilar Equational Abstraction). Given a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ and a set G of extra equations, an equational abstraction $\mathcal{R}/G$ is a *bisimilar equational abstraction* iff for any states $t_1, t_2, t_3 \in \mathcal{T}_{\Sigma/E,\mathsf{State}}$:

$$t_1 \longrightarrow_{R,E} t_2 \ \wedge\ t_1 =_{E\cup G} t_3 \implies (\exists t_4 \in \mathcal{T}_{\Sigma/E,\mathsf{State}})\ t_3 \longrightarrow_{R,E} t_4 \ \wedge\ t_2 =_{E\cup G} t_4.$$

Notice that for a bisimilar equational abstraction $\mathcal{R}/G$, the equivalence relation $=_{E\cup G}$ is indeed a bisimulation for $\mathcal{R}$ with respect to $\longrightarrow_{R,E}$, since $=_{E\cup G}$ is symmetric.

▶ **Theorem 16.** *Given a topmost rewrite theory $\mathcal{R} = (\Sigma, E, R)$ and a finite set AP of state propositions defined by the equations E, if $\mathcal{R}/G$ is a bisimilar equational abstraction, then $H_G = \{([t]_E, [t]_{E\cup G}) \mid t \in \mathcal{T}_\Sigma(\mathcal{X})_{\mathsf{State}}\}$ is a bisimulation between $\mathcal{N}_{\mathcal{R}}^{AP}$ and $\mathcal{N}_{\mathcal{R}/G}^{AP}$.*

Proof. We only need to prove the H_G is a simulation between $\mathcal{N}_{\mathcal{R}/G}^{AP}$ and $\mathcal{N}_{\mathcal{R}}^{AP}$. For terms $u \in \mathcal{T}_{\Sigma/G}(\mathcal{X})$ and $t \in \mathcal{T}_\Sigma(\mathcal{X})$, suppose $u =_{E\cup G} t$ and $u \rightsquigarrow_{\sigma,R,E\cup G} u'$, i.e., $\sigma u \longrightarrow_{R,E\cup G} u'$. Since $\sigma u =_{E\cup G} \sigma t$, by definition of bisimilar equational abstractions, there exists a term $t' \in \mathcal{T}_{\Sigma/E}$ such that $\sigma t \longrightarrow_{R,E} t'$ and $u' =_{E\cup G} t'$. Therefore, by completeness of narrowing, there exists a substitution σ' such that $t \rightsquigarrow_{\sigma',R,E} t'$. ◀

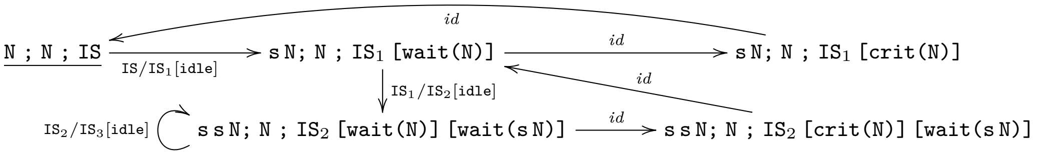
N ; N ; IS
id
IS/IS$_1$ [idle]
s N; N ; IS$_1$ [wait(N)]
id
s N; N ; IS$_1$ [crit(N)]
IS$_1$/IS$_2$ [idle]
id
IS$_2$/IS$_3$ [idle]
s s N; N ; IS$_2$ [wait(N)] [wait(s N)]
id
s s N; N ; IS$_2$ [crit(N)] [wait(s N)]

Figure 5 An abstract folded logical transition system for the bakery example.

A bisimilar equational abstraction with *topmost equations* of the form $t = t'$ with $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})_{\mathsf{State}}$ can be easily identified by checking critical pairs between the left-hand sides of the rules and both sides of the equations in the equational abstraction, and checking that application of an equation does not interfere with the application of a rewrite rule. This process can easily be automated in a similar way to the existing Maude coherence checker [13], which checks similar (but slightly different) conditions between equations and rules.

▶ **Theorem 17** (Necessary/Sufficient Conditions for Bisimilarity). *Given a topmost rewrite theory $\mathcal{R}$ and an equational abstraction $\mathcal{R}/G$ for a set G of* topmost *equations, $\mathcal{R}/G$ is a bisimilar equational abstraction iff for each rule $l \longrightarrow r$ and each $u = v \in G$ or $v = u \in G$:*

$$\sigma \in CSU_E(l = u) \implies \big(\exists \theta : \mathcal{X} \to \mathcal{T}_\Sigma(\mathcal{X})\big)\ \sigma v =_E \theta l \ \wedge\ \sigma r =_{E\cup G} \theta r. \qquad (*)$$

Proof. (If) Suppose that $t_1 \longrightarrow_{R,E} t_2$ and $t_1 =_{E\cup G} t_3$. If $=^k_{G/E}$ denotes k applications of equations in G modulo E, then $t_1 =^n_{G/E} t_3$ for some $n \in \mathbb{N}$. We prove by induction on the number n that $(\exists t_4)\ t_3 \longrightarrow_{R,E} t_4$ and $t_2 =_{E\cup G} t_4$. When $n = 0$, it is immediate because $t_1 =_E t_3$ and then $t_3 \longrightarrow_{R,E} t_2$. For $n > 0$, assume that if $t_1 =^n_{G/E} t_3'$ for any $t_3' \in \mathcal{T}_{\Sigma/E,\mathsf{State}}$, there exists $t_4' \in \mathcal{T}_{\Sigma/E,\mathsf{State}}$ such that $t_3' \longrightarrow_{R,E} t_4'$ and $t_2 =_{E\cup G} t_4'$. If $t_1 =^n_{G/E} t_3' =^1_{G/E} t_3$, then $t_3' \longrightarrow_{R,E} t_4'$ and by using the critical pair condition in the statement, $(\exists t_4)\ t_3 \longrightarrow_{R,E} t_4$ and $t_4' =_{E\cup G} t_4$. Finally, we have that $t_2 =_{E\cup G} t_4' =_{E\cup G} t_4$ and the conclusion follows.
(Only if) The property $t_1 \longrightarrow_{R,E} t_2 \ \wedge\ t_1 =_{E\cup G} t_3 \implies (\exists t_4)\ t_3 \longrightarrow_{R,E} t_4 \ \wedge\ t_2 =_{E\cup G} t_4$ must be satisfied for a term t_3 that has only one application of the equations in G, i.e., $t_1 =^1_{G/E} t_3$. And such a case is indeed represented by the conditions of the statement. ◀

The reason why only topmost equations are allowed for bisimilar equational abstractions is to avoid problems caused by repeated variables in transition rules. For instance, consider $R = \{f(X, X) \longrightarrow h(X)\}$, $E = \emptyset$ and $G = \{a = b\}$. This topmost rewrite theory satisfies the condition of the previous theorem except G being topmost. Then, given the term $f(a, a)$, $f(b, a) =_G f(a, a)$ but now $f(b, a)$ cannot be rewritten with R.

For our bakery example, we can obtain a bisimilar equational abstraction of the folded transition system by restricting the abstraction only to the following equation, which intuitively collapses *extra* waiting processes that does not introduce any new behaviors:

```
eq (s s s L M) ; M ; PS0 [wait(s L M)] [wait(s s L M)]
 =   (s s L M) ; M ; PS0 [wait(s L M)] .
```

▶ **Lemma 18.** *The above equation satisfies the bisimilarity conditions (*) in Theorem 17.*

Proof. If we consider the rule `[wake]: N ; M ; [idle] PS => (s N) ; M ; [wait(N)] PS`, then $CSU_E(l = u)$ has the single E-unifier $\sigma = \{\mathtt{PS} \mapsto \mathtt{PS}_1\,\mathtt{[wait(s\,M\,L)]\,[wait(s\,s\,M\,L)]},\ \mathtt{N} \mapsto \mathtt{s\,s\,s\,M\,L},\ \mathtt{PS}_0 \mapsto \mathtt{PS}_1\,\mathtt{[idle]}\}$, where E denotes the equational axioms. Then, $\sigma v =_E \theta l$ and $\sigma r =_{E\cup G} \theta r$, for the substitution $\theta = \{\mathtt{N} \mapsto \mathtt{s\,s\,M\,L},\ \mathtt{PS} \mapsto \mathtt{PS}_1\,\mathtt{[wait(s\,M\,L)]}\}$. For the other direction of the equation, $CSU_E(l = v) = \big\{\sigma' = \{\mathtt{PS} \mapsto \mathtt{PS}_2\,\mathtt{[wait(s\,M\,L)]},\ \mathtt{PS}_0 \mapsto \mathtt{PS}_2\,\mathtt{[idle]},\ \mathtt{N} \mapsto \mathtt{s\,s\,M\,L}\}\big\}$, and for the substitution $\theta' = \{\mathtt{N} \mapsto \mathtt{s\,s\,s\,M\,L},\ \mathtt{PS} \mapsto \mathtt{PS}_2\,\mathtt{[wait(s\,M\,L)]\ [wait(s\,s\,M\,L)]}\}$, we have $\sigma' u =_E \theta' l$ and $\sigma' r =_{E\cup G} \theta' r$. The cases for the other rules are similar. ◀

Therefore, in order to model check the invariant $\square$`ex?` from the initial pattern `N ; N ; IS` for an unbounded number of processes, we can then construct the *finite* abstract folded logical transition system from the given initial pattern displayed in Figure 5, where any counterexamples found are *not* spurious by Theorems 10 and 16.

4 Logical Bounded LTL Model Checking with Folding

We have shown that an infinite *logical* state space can be reduced to a finite state space using folding abstractions and equational abstractions. Although we can always achieve a finite logical state space using a *trivial* equational abstraction that collapses every state into a single state, the interesting case is obtaining *bisimilar* equational abstractions that produce a finite logical state space. However, we cannot ensure *a priori* whether such an abstract logical state space is finite or not, since it is in general undecidable for many infinite-state systems. Therefore, we introduce a logical bounded model checking (LBMC) method for verifying LTL properties, which provides an under-approximation of a logical state space.

In LBMC, we construct a *k-step folding abstraction* of $\mathcal{K}$ whose states are reachable in k-steps from a set of initial states $I \subseteq S$. Such a depth k is iteratively incremented until a certain bound or until reaching a fixed-point if it exists.

▶ **Definition 19.** Given $\mathcal{K} = (S, AP, \mathcal{L}, \longrightarrow_{\mathcal{K}})$, a folding preorder $\preccurlyeq \subseteq S^2$, and a set of initial states $I \subseteq S$, the *k-step folding abstraction* of $\mathcal{K}$ from I is the Kripke structure

$$\mathcal{R}each^{\preccurlyeq,k}_{\mathcal{K}}(I) = (Post^{\leq k}_{\mathcal{K}\preccurlyeq}(I),\ AP,\ \mathcal{L},\ \longrightarrow_{\mathcal{R}each^{\preccurlyeq,k}_{\mathcal{K}}(I)}),$$

where $Post^{\leq k}_{\mathcal{K}\preccurlyeq}(I) = \bigcup_{0\leq i\leq k} Post^{i}_{\mathcal{K}\preccurlyeq}(I)$ and $\longrightarrow_{\mathcal{R}each^{\preccurlyeq,k}_{\mathcal{K}}(I)} = \bigcup_{0\leq i\leq k} \longrightarrow^{\preccurlyeq}_{\mathcal{K},i}$.

For a (∞-step) folding abstraction $\mathcal{R}each^{\preccurlyeq}_{\mathcal{K}}(I)$ we can easily see that if its state set $Post^{*}_{\mathcal{K}\preccurlyeq}(I)$ is finite, there exists $n \in \mathbb{N}$ such that $\mathcal{R}each^{\preccurlyeq,j}_{\mathcal{K}}(I) = \mathcal{R}each^{\preccurlyeq}_{\mathcal{K}}(I)$ for any $j \geq n$ by definition. Therefore, unlike typical bounded model checking methods (e.g., [3]), our folding-based method can easily detect if $\mathcal{R}each^{\preccurlyeq,n}_{\mathcal{K}}(I)$ is *complete* or not.

The LBMC of a logical Kripke structure $\mathcal{N}$ with a set of initial states I and a folding preorder $\preccurlyeq$ consists in model checking $\mathcal{N}(I)^{\preccurlyeq}_{i} = \mathcal{R}each^{\preccurlyeq;i}_{\mathcal{N}}(I)$ for each $i \in \mathbb{N}$, iteratively from 0 until one of the following termination conditions holds: (i) $\mathcal{N}(I)^{\preccurlyeq}_{i}$ is complete (a fixpoint is found), (ii) a counterexample is found in $\mathcal{N}(I)^{\preccurlyeq}_{i}$, or (iii) i is greater than a given maximum bound n. The LBMC algorithm for an LTL formula φ is briefly described as follows:

1. Apply a standard explicit-state LTL model checking algorithm to verify φ on $\mathcal{N}(I)^{\preccurlyeq}_{k}$. If a counterexample of φ is found in $\mathcal{N}(I)^{\preccurlyeq}_{k}$, stop and return the counterexample.
2. Suppose that there is *no* counterexample of φ in $\mathcal{N}(I)^{\preccurlyeq}_{k}$.
   - a. If $k \geq n$, stop and report that $\mathcal{N}$ does not violate φ until the current bound k.
   - b. Otherwise, generate $\mathcal{N}(I)^{\preccurlyeq}_{k+1}$ with the next bound $k+1$:
     - i. If $\mathcal{N}(I)^{\preccurlyeq}_{k+1}$ is identical to $\mathcal{N}(I)^{\preccurlyeq}_{k}$, that is, $\mathcal{N}(I)^{\preccurlyeq}_{k}$ is complete, return *true*;
     - ii. Otherwise, increment the depth-bound k by 1 and go to Step 1.

If the LBMC algorithm returns a counterexample, there are three possibilities according to the underlying folding preorder. If $\preccurlyeq$ is the E-renaming equivalence $\approx_E$ or φ is an invariant, it is an actual counterexample in $\mathcal{N}$. If $\preccurlyeq$ is the E-subsumption $\preccurlyeq_E$ and φ is a general LTL formula, it may be a spurious counterexample. Of course, if an equation abstraction has been applied, it is a real counterexample in $\mathcal{N}$ only for a bisimilar equational abstraction. Note that the above LBMC algorithm can easily be extended to guarantee the faithfulness for *safety* LTL properties as explained in Section 3.2.

5 The Maude LTL Logical Model Checker and Examples

This section illustrates the Maude LTL logical model checker (LMC) tool with two examples. This tool uses the existing narrowing framework in Full Maude to compute *narrowing* $\rightsquigarrow_{\sigma,R,E}$ [12]. However, for efficiency reasons, the core algorithms for the folding graph construction and the LTL model checking are implemented at the C++ level within the Maude system. For the LBMC algorithm, we apply an on-the-fly technique to reuse the previously generated states for the next step. The Maude LTL LMC tool and a number of other examples can be found in `http://formal.cs.illinois.edu/kbae/lmc`.

Our tool provides the following two commands for logical model checking an LTL formula φ from an initial state t with the maximum bound $n \in \mathbb{N}$:

`(lmc [`n`]` t `|=` φ `.)` and `(lfmc [`n`]` t `|=` φ `.)`

This bound n limits the depth of the k-step folding graph $\mathcal{R}each^{\preccurlyeq,k}_{\mathcal{N}^{AP}_{\mathcal{R}}}([t]_E)$ from an initial state $[t]_E \in \mathcal{N}^{AP}_{\mathcal{R}}$. Each command uses a different folding relation $\preccurlyeq$: the *renaming equivalence* $\approx_E$ for the `lmc` command, and the *subsumption* $\preccurlyeq_E$ for the `lfmc` command. If a bound n is not specified in the command, infinity is considered as the bound.

5.1 The Bakery Algorithm Revisited

The following command partially verifies that the mutual execution $\Box$`ex?` is satisfied from any initial state with the pattern `N ; N ; IS:ProcIdleSet` within the bound 10:

```
Maude> (lmc [10] N:Nat ; N:Nat ; IS:ProcIdleSet |= [] ex? .)
logical model check in BAKERY-SATISFACTION :
  N:Nat ; N:Nat ; IS:ProcIdleSet |= [] ex?
result:
  no counterexample found within bound 10
```

This model checking command does not terminate if the bound is not specified, since $\approx_E$ is not strong enough to collapse the reachable transition system to a finite one. The bound should be specified to ensure the termination even with $\preccurlyeq_E$, since, as already shown in Figure 4, for such a logical initial state the folding logical approximation is infinite:

```
Maude> (lfmc [50] N:Nat ; N:Nat ; IS:ProcIdleSet |= [] ex? .)
logical folding model check in BAKERY-SATISFACTION :
  N:Nat ; N:Nat ; IS:ProcIdleSet |= [] ex?
result:
  no counterexample found within bound 50
```

Instead, when the subsumption $\preccurlyeq_E$ is applied, with the bisimilar equational abstraction shown in Section 3.3, the mutual exclusion property $\Box$`ex?` can be verified from the initial pattern `N ; N ; IS:ProcIdleSet` as follows,[1] where, as shown in Figure 5, five logical states are generated in less than one second on an Intel Core i5 2.4 GHz with 4GB RAM:

```
Maude> (lfmc N:Nat ; N:Nat ; IS:ProcIdleSet |= [] ex? .)
logical folding model check in BAKERY-SATISFACTION-ABS :
  N:Nat ; N:Nat ; IS:ProcIdleSet |= [] ex?
result:
  true
```

[1] Note that the module `BAKERY-SATISFACTION-ABS` extends the previous module `BAKERY-SATISFACTION` with the abstraction equation in Section 3.3.

```
l0:   repeat
l1:     flag[i] := 1
l2:     while turn ≠ i do
          if flag[turn] = 0 then turn := i
l3:     flag[i] := 2
l4:     for j ≠ i do
          if flag[j] = 2 then goto l1
crit:   /* critical region */
l5:     flag[i] := 0
      forever
```

Figure 6 The Dijkstra's Mutual Exclusion Algorithm (for a process i) [27]

5.2 Dijkstra's Mutual Exclusion Algorithm

This section illustrates a topmost rewrite theory with another mutual exclusion algorithm for an arbitrary number of processes. Dijkstra's algorithm [27] considers n processes with $n \geq 2$, and two shared variables: (i) $\mathit{flag}[1 \ldots n]$ is an array of values $\{0, 1, 2\}$ for each process $1 \leq i \leq n$, and (ii) *turn* is an integer between 1 and n. The behavior of this algorithm is summarized by the pseudo code in Figure 6.

We represent a state of this system as a multiset of triples $< \{f_1, p_1, t_1\} \cdots \{f_k, p_k, t_k\} >$, where each $\{f_i, p_i, t_i\}$ represents a process with f_i a value of $\mathit{flag}[i]$, p_i a program counter, and t_i a turn specifier that can be either `on` (i.e., $\mathit{turn} = i$) or `off` (i.e., $\mathit{turn} \neq i$). Only one process can be turned on at a time. The behavior of this system is then specified by the following topmost rewrite rules, where `PS` stands for the remaining set of precesses, and `WAITPS` stands for a multiset of processes whose flag is either 0 or 1:

```
rl [l1] : < {F,l0,T} PS >                => < {1,l1,T} PS > .
rl [l2] : < {F,l1,off} {0,S,on} PS >     => < {F,l1,on} {0,S,off} PS > .
rl [l2']: < {F,l1,on} PS >               => < {F,l2,on} PS > .
rl [l3] : < {F,l2,T} PS >                => < {2,l3,T} PS > .
rl [l4] : < {F,l3,T} {2,S,T'} PS >       => < {1,l1,T} {2,S,T'} PS > .
rl [l4']: < {F,l3,T} WAITPS >            => < {F,crit,T} WAITPS > .
rl [l5] : < {F,crit,T} PS >              => < {0,l5,T} PS > .
rl [l0] : < {F,l5,T} PS >                => < {F,l0,T} PS > .
```

Similar to the Bakery example, the mutual exclusion property of a single state can be specified by the atomic proposition `ex?`, defined by the following equations, where the logical variable `NCPS` stands for a set of processes whose program counter is *not* `crit`:

```
eq < NCPS > |= ex? = true .
eq < {F,crit,T} NCPS > |= ex? = true .
eq < {F,crit,T} {F',crit,T'} PS > |= ex? = false .
```

This system is infinite-state since the number of processes is unbounded. As a result, if the logical variable `IS` denotes a set of processes with flag 0 and program counter `l0`, the reachable logical state space from the pattern `< IS:InitProcSet >` is infinite even with the subsumption $\preccurlyeq_E$. However, we can obtain a *finite* bisimilar equational abstraction by adding the following topmost equations,[2] which satisfy the conditions (*) in Theorem 17:

```
eq < {F,l1,off} {F,l1,off} PS > = < {F,l1,off} PS > .
eq < {F,l5,off} {F,l5,off} PS > = < {F,l5,off} PS > .
```

[2] These equations G do *not* satisfy the finite variant property (see the conditions on [17]). However, all the *reachable* logical states from the given initial state `< IS >` have a finite set of most general G-variants, which is enough to have a finitary G-unification procedure for the *reachable* logical state space.

Then, the mutual exclusion $\Box$`ex?` can be verified from the pattern `< IS:InitProcSet >` that represents an arbitrary number of processes by the following command, where 42 logical spaces are generated in less than 6 second on the same machine:

```
Maude> (lfmc < IS:InitProcSet > |= [] ex? .)
logical folding model check in DIJKSTRA-MUTEX-SATISFACTION-ABS:
  < IS:InitProcSet > |= [] ex?
result:
  true
```

6 Conclusions

Using narrowing to model check LTL formulas on infinite-state systems is an intriguing symbolic method proposed in [16], whose practical effectiveness has required finding new methods to solve several open problems —such as Problems (1)–(3) in Section 1— and demonstrating its effectiveness in practice by supporting tools and examples. This paper has presented several new methods solving, or substantially improving, many of these difficulties, and the new Maude LTL logical model checker supporting these new techniques. We have also shown the effectiveness of the tool in verifying two nontrivial examples.

As usual much work remains ahead. Although the execution times shown in examples are quite reasonable, the tool's efficiency can be substantially improved by systematically exploiting the folding variant narrowing and unification features implemented at the C++ level in the upcoming new release of Maude. We plan to do this in the near future. Also, although the tool implementation is reasonably mature and has been tested on a collection of nontrivial examples, more experimentation is needed to increase its performance and illustrate its use on a wider set of applications. Another promising research direction is combining narrowing and SMT solving, using the *rewriting modulo SMT* ideas [33].

Acknowledgments. This work has been partially supported by NSF Grant CCF 09-05584, and for Santiago Escobar by the EU (FEDER) and the Spanish MEC/MICINN under grant TIN 2010-21062-C02-02 and by Generalitat Valenciana PROMETEO2011/052.

References

1 P. Abdulla, B. Jonsson, P. Mahata, and J. d'Orso. Regular tree model checking. In *CAV*, pages 452–466. Springer, 2002.

2 C. Baier and J. P. Katoen. *Principles of Model Checking*. The MIT Press, 2007.

3 A. Biere, A. Cimatti, E.M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in computers*, 58:117–148, 2003.

4 A. Bouajjani and J. Esparza. Rewriting models of boolean programs. *Term Rewriting and Applications*, pages 136–150, 2006.

5 A. Bouajjani and T. Touili. Widening techniques for regular tree model checking. *International Journal on Software Tools for Technology Transfer*, 14(2):145–165, 2012.

6 T. Bultan, R. Gerber, and W. Pugh. Symbolic model checking of infinite state systems using presburger arithmetic. In *CAV*, pages 400–411. Springer, 1997.

7 O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. *Handbook of Process algebra*, pages 545–623, 2001.

8 E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 2001.

9 M. Clavel, F. Durán, S. Eker, J. Meseguer, P. Lincoln, N. Martí-Oliet, and C. Talcott. *All About Maude*, volume 4350 of *LNCS*. Springer, 2007.

10 H. Comon-Lundh and S. Delaune. The finite variant property: How to get rid of some algebraic properties. *Term Rewriting and Applications*, pages 294–307, 2005.

11 G. Delzanno and A. Podelski. Constraint-based deductive model checking. *STTT*, 3, 2001.

12 F. Durán, S. Eker, S. Escobar, J. Meseguer, and C. Talcott. Variants, unification, narrowing, and symbolic reachability in maude 2.6. In *RTA*, volume 10, pages 31–40, 2011.

13 F. Durán and J. Meseguer. A maude coherence checker tool for conditional order-sorted rewrite theories. In *Rewriting Logic and Its Applications*. Springer, 2010.

14 E. A. Emerson and K. S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *Logic in Computer Science*, pages 70–80. IEEE, 1998.

15 S. Escobar, C. Meadows, and J. Meseguer. Maude-NPA: cryptographic protocol analysis modulo equational properties. In *Foundations of Security Analysis and Design V*, volume 5705 of *LNCS*, pages 1–50. Springer, 2009.

16 S. Escobar and J. Meseguer. Symbolic model checking of infinite-state systems using narrowing. In *RTA*, pages 153–168, 2007.

17 S. Escobar, R. Sasse, and J. Meseguer. Folding variant narrowing and optimal variant termination. *J. Algebraic and Logic Programming*, 81:898–928, 2012.

18 A. Farzan and J. Meseguer. State space reduction of rewrite theories using invisible transitions. *Algebraic Methodology and Software Technology*, pages 142–157, 2006.

19 A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.

20 F. Fioravanti, A. Pettorossi, M. Proietti, and V. Senni. Program specialization for verifying infinite state systems: An experimental evaluation. In *Logic-Based Program Synthesis and Transformation*, volume 6564 of *LNCS*, pages 164–183. Springer, 2010.

21 T. Genet and V. Rusu. Equational approximations for tree automata completion. *Journal of Symbolic Computation*, 45(5):574–597, 2010.

22 T. Genet and V. Tong. Reachability analysis of term rewriting systems with timbuk. In *Logic for Programming, Artificial Intelligence, and Reasoning*. Springer, 2001.

23 J. M. Hullot. Canonical forms and unification. In *CADE*, LNCS vol. 87. Springer, 1980.

24 J. P. Jouannaud, C. Kirchner, and H. Kirchner. Incremental construction of unification algorithms in equational theories. In *Proc. ICALP*, volume 154 of *LNCS*. Springer, 1983.

25 Y. Kesten and A. Pnueli. Control and data abstraction: The cornerstones of practical formal verification. *STTT*, 2(4):328–342, 2000.

26 O. Kupferman and M.Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

27 Nancy Ann Lynch. *Distributed algorithms*. Morgan Kaufmann, 1996.

28 J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theor. Comput. Sci.*, 96(1):73–155, 1992.

29 J. Meseguer. Membership algebra as a logical framework for equational specification. In *WADT*, volume 1376 of *LNCS*, pages 18–61. Springer, 1997.

30 J. Meseguer, M. Palomino, and N. Martí-Oliet. Equational abstractions. *Theor. Comput. Sci.*, 403(2-3):239–264, 2008.

31 J. Meseguer and P. Thati. Symbolic reachability analysis using narrowing and its application to verification of cryptographic protocols. *Higher-Order and Symbolic Computation*, 20(1-2):123–160, 2007.

32 H. Ohsaki, H. Seki, and T. Takai. Recognizing boolean closed a-tree languages with membership conditional rewriting mechanism. In *RTA*, pages 483–498. Springer, 2003.

33 C. Rocha. *Symbolic Reachability Analysis for Rewrite Theories*. PhD thesis, University of Illinois at Urbana-Champaign, 2012.

34 A. Vardhan, K. Sen, M. Viswanathan, and G. Agha. Using language inference to verify omega-regular properties. *TACAS*, pages 45–60, 2005.