Combining Predicate Abstraction with Fixpoint Approximations

Tuba Yavuz

University of Florida,
tuba@ece.ufl.edu

Abstract. In this paper we consider combining two techniques that have been effective in analyzing infinite-state systems: predicate abstraction and fixpoint approximations. Using a carefully crafted model of Airport Ground Network Control, we show that when predicate abstraction in a CEGAR loop fails to verify temporal logic properties of an infinite-state transition system, a combination of predicate abstraction with fixpoint approximations may provide improved performance for both safety and liveness property verification.

Keywords: predicate abstraction, widening, model checking

1 Introduction

State-explosion is an inherent problem in model checking. Every model checking tool - no matter how optimized - will report or demonstrate one of the following for systems that push its limits: out of memory error, non-convergence, or inconclusive result. As the target systems of interest (hardware, software, or biological systems) grow in terms of complexity, and consequently in size, a great deal of manual effort is spent on verification engineering to produce usable results. We admit that this effort will always be needed. However, we also think that hybrid approaches should be employed to push the limits for automated verification.

Abstract interpretation framework [6] provides a theoretical basis for sound verification of finite as well as infinite-state systems. Two major elements of this framework are abstraction and approximation. Abstraction defines a mapping between a concrete domain and an abstract domain (less precise) in a conservative way so that when a property is satisfied for an abstract state the property also holds for the concrete states that map to the abstract state. Approximation, on the other hand, works on values in the same domain and provides a lower or an upper bound. Abstraction is a way to deal with the state-explosion problem whereas approximation is a way to achieve convergence and hence potentially a conclusive result. When an infinite-state system is considered there are three basic approaches that can be employed: pure abstraction, pure approximation[1], and a combination of abstraction and approximation.

[1] Assuming the logic that describes the system is decidable.

The most popular abstraction technique is predicate abstraction [9], in which the abstract domain consists of a combination of valuations of Boolean variables that represent truth values of a fixed set of predicates on the variables from the concrete system. Since it is difficult to come up with the right set of predicates that would yield a precise analysis, predicate abstraction has been combined with the counter-example guided abstraction refinement (CEGAR) framework. Predicate abstraction requires computing a quantifier-free version of the transformed system and, hence, potentially involves an exponential number of queries to the underlying SMT solver.

A widely used approximation technique is *widening*. The widening operator takes two states belonging to the same domain and computes an over-approximation of the two. A key point of the widening operator is the guarantee for stabilizing an increasing chain after a finite number of steps. So one can apply the widening operator to the iterates of a non-converging fixpoint computation and achieve convergence, where the last iterate is an over-approximation of the actual fixpoint. In this paper we use an implementation of the widening operator for convex polyhedra [8] that is used in the infinite-state model checker Action Language Verifier (ALV) [18]. ALV uses fixpoint approximations to check whether a CTL property is satisfied by an infinite-state system [2].

In [7] it is demonstrated that both model checking and automated testing can benefit from a combination of carefully designed abstractions and approximations that improve precision of the analysis. In this paper, we take a modest step by combining predicate abstraction with widening for infinite-state systems described in terms of Presburger arithmetic. Our approach requires the user to provide the set of predicates to be considered. *In our approach only the variables that are involved in the predicates are abstracted and all other variables are preserved in their concrete domains.* We implemented the combined approach by extending the Action Language Verifier (ALV) [18] with automated predicate abstraction capability. We show the need for such a combined approach through a specially crafted infinite-state model that requires fixpoint approximation. Our experimental results show that combining the two techniques can provide improved performance for safety as well as liveness property specification.

The rest of the paper is organized as follows. We first present the basic definitions and key results of the two approaches, approximate fixpoint computations and predicate abstraction in the context of CTL model checking, in Section 2. Section 3 presents the hybrid approach and demonstrates soundness of combining the two techniques. Section 4 presents the experimental results. Section 5 discusses related work and Section 6 concludes with directions for future work.

2 Preliminaries

In this paper, we consider transition systems that are described in terms of boolean and unbounded integer variables.

Definition 1. *An infinite-state transition system is described by a Kripke structure $T = (S, I, R, V)$, where S, I, R, and V denote the state space, set of ini-*

tial states, the transition relation, and the set of state variables, respectively. $V = V_{bool} \cup V_{int}$ *such that* $S \subseteq \mathcal{B}^{|V_{bool}|} \times \mathcal{Z}^{|V_{int}|}$, $I \subseteq S$, *and* $R \subseteq S \times S$.

Definition 2. *Given a Kripke structure,* $T = (S, I, R, V)$ *and a set of states* $A \subseteq S$, *the post-image operator,* $post[R](A)$, *computes the set of states that can be reached from the states in* A *in one step:*

$$post[R](A) = \{b \mid a \in A \ \wedge \ (a, b) \in R\}.$$

Similarly, the pre-image operator, $pre[R](A)$, *computes the set of states that can reach the states in* A *in one step:*

$$pre[R](A) = \{b \mid a \in A \ \wedge \ (b, a) \in R\}.$$

Model Checking via Fixpoint Approximations. Symbolic Computation-Tree Logic (CTL) model checking algorithms decide whether a given Kripke structure, $T = (S, I, R, V)$, satisfies a given CTL correctness property, f, by checking whether $I \subseteq [\![f]\!]_T$, where $[\![f]\!]_T$ denotes the set of states that satisfy f in T. Most CTL operators have either least fixpoint (EU, AU) or greatest fixpoint (EG, AG) characterizations in terms of the pre-image operator.

Variables	s, t, a_1, a_2, z: integer
	$pc1$, $pc2$: think, try, cs
Initial State:	$s = t \wedge pc_1 = think \wedge pc_2 = think$
Transitions:	
r_i^{try}	$\equiv pc_i = think \wedge a_i' = t \wedge t' = t + 1 \wedge pc_i' = try$
r_i^{cs}	$\equiv pc_i = try \wedge s \geq a_i \wedge z' = z + 1 \wedge pc_i' = cs$
r_i^{think}	$\equiv pc_i = cs \wedge s' = s + 1 \wedge z' = z - 1 \wedge pc_i' = think$
Transition Relation:	$\bigvee_{i=1,2} r_i^{try} \vee r_i^{cs} \vee r_i^{think}$

Fig. 1. The ticket mutual exclusion algorithm for two processes. Variable z is an addition to demonstrate the merits of the proposed approach.

Symbolic CTL model checking for infinite-state systems may not converge. Consider the so-called ticket mutual exclusion model for two processes [1] given in Figure 1. Each process gets a ticket number before attempting to enter the critical section. There are two global integer variables, t and s, that show the next ticket value that will be available to obtain and the upper bound for tickets that are eligible to enter the critical section, respectively. Local variable a_i represents the ticket value held by process i. We added variable z to model an update in the critical region. It turns out that checking $AG(z \leq 1)$ for this model does not terminate.

One way is to compute an over or an under approximation to the fixpoint computations as proposed in [2] and check $I \subseteq [\![f]\!]_T^-$, i.e., check whether all initial

states in T satisfy an under-approximation (denoted by superscript $-$) of the correctness property or check $I \cap [\![\neg f]\!]^+_T \neq \emptyset$, i.e., check whether no initial state satisfies an over-approximation of the negated correctness property. If so, the model checker certifies that the property is satisfied. Otherwise, no conclusions can be made without further analysis.

The key in approximating a fixpoint computation is the availability of over-approximating and under-approximating operators. So we give the basic definitions and a brief explanation here and refer the reader to [8, 2] for technical details on the implementation of these operators for Presburger arithmetic.

Definition 3. *Given a complete lattice* $(L, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$, $\triangle : L \times L \to L$, *is a widening operator iff*

- $\forall x, y \in L.\ x \sqcup y \sqsubseteq x \triangle y$,
- *For all increasing chains* $x_0 \sqsubseteq x_1 \sqsubseteq ...x_n$ *in* L, *the increasing chain* $y_0 = x_0, ..., y_{n+1} = y_n \triangle x_{n+1}, ...$ *is not strictly increasing, i.e., stabilizes after a number of terms.*

Definition 4. *Given a complete lattice* $(L, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$, $\nabla : L \times L \to L$, *is a dual of the widening operator iff*

- $\forall x, y \in L. x \nabla y \sqsubseteq x \sqcap y$,
- *For all decreasing chains* $x_0 \sqsupseteq x_1 \sqsupseteq ...x_n$ *in* L, *the decreasing chain* $y_0 = x_0, ..., y_{n+1} = y_n \nabla x_{n+1}, ...$ *is not strictly decreasing, i.e., stabilizes after a number of terms.*

The approximation of individual temporal operators in a CTL formula is decided recursively based on the type of approximation to be achieved and whether the operator is preceded by a negation. The over-approximation can be computed using the widening operator for least fixpoint characterizations and terminating the fixpoint iteration after a finite number of steps for greatest fixpoint characterizations. The under-approximation can be computed using the dual of the widening operator for the greatest fixpoint characterizations and terminating the fixpoint iteration after a finite number of steps for the least fixpoint characterizations. Another heuristic that is used in approximate symbolic model checking is to compute an over-approximation (denoted by superscript $+$) of the set of reachable states ($(\mu Z.I \vee post[R](Z))^+$), a least fixpoint characterization, and to restrict all the fixpoint computations within this set.

Lemma 1. *Given an infinite-state transition system* $T = (S, I, R, V)$ *and* $T^+ = ((\mu Z.I \vee post[R](Z))^+, I, R, V)$, *and a temporal property* f, *the conclusive results obtained using fixpoint approximations for the temporal operators and the approximate set of reachable states are sound, i.e.,* $(I \subseteq [\![f]\!]^-_{T^+} \ \vee \ I \cap [\![\neg f]\!]^+_{T^+} = \emptyset) \to T \models f$ *(see [2] for the proof).*

So for the example model in Figure 1, an over-approximation to $EF(z > 1)$, the negation of the correctness property, is computed using the widening operator. Based on the implementation of the widening operator in [18], it turns out that the initial states do not intersect with $[\![EF(z > 1)]\!]^+_{ticket2}$ and hence the model satisfies $AG(z \leq 1)$.

Abstract Model Checking and Predicate Abstraction.

Definition 5. *Let φ denote a set of predicates over integer variables. Let φ_i denote a predicate in φ and b_i denote the boolean variable that corresponds to φ_i. $\bar{\varphi}$ represents an ordered sequence (from index 1 to $|\varphi|$) of predicates in φ. The set of variables that appear in φ is denoted by $V(\varphi)$. Let φ' denote the set of next state predicates obtained from φ by replacing variables in each predicate φ_i with their primed versions. Let b denote the set of b_i that corresponds to each φ_i. Let $V_\sharp = V_\natural \cup b \setminus V(\varphi)$, where $V_\natural$ denotes the set of variables in the concrete model.*

Abstracting states. A concrete state $s^\natural$ is predicate abstracted using a mapping function α via a set of predicates φ by introducing a predicate boolean variable b_i that represents predicate φ_i and existentially quantifying the concrete variables $V(\varphi)$ that appear in the predicates:

$$\alpha(s^\natural) = \exists V(\varphi).(s^\natural \ \wedge \ \bigwedge_{i=1}^{|\varphi|} \varphi_i \iff b_i). \tag{1}$$

Concretization of abstract states. An abstract state $s^\sharp$ is mapped back to all the concrete states it represents by replacing each predicate boolean variable b_i with the corresponding predicate φ_i:

$$\gamma(s^\sharp) = s^\sharp[\bar{\varphi}/\bar{b}] \tag{2}$$

Abstraction function α provides a safe approximation for states:

Lemma 2. *(α, γ), as defined in Equations 1 and 2, defines a Galois connection, i.e., α and γ are monotonic functions and $s^\natural \subseteq \gamma(\alpha(s^\natural))$ and $\alpha(\gamma(s^\sharp)) = s^\sharp$ (see the Appendix for the proof).*

A concrete transition system can be conservatively approximated by an abstract transition system through a simulation relation or a surjective mapping function involving the respective state spaces:

Definition 6. *(Existential Abstraction) Given transition systems $T_1 = (S_1, I_1, R_1, V_1)$ and $T_2 = (S_2, I_2, R_2, V_2)$, T_2 approximates T_1 (denoted $T_1 \sqsubseteq_h T_2$) iff*

- *$\exists s_1.(h(s_1) = s_2 \ \wedge \ s_1 \in I_1)$ implies $s_2 \in I_2$,*
- *$\exists s_1, s_1'.(h(s_1) = s_2 \ \wedge \ h(s_1') = s_2' \ \wedge \ (s_1, s_1') \in R_1)$ implies $(s_2, s_2') \in R_2$,*

where h is a surjective function from S_1 to S_2.

It is a known [13] fact that one can use a Galois connection (α, γ) to construct an approximate transition system. Basically, α is used as the mapping function and γ is used to map properties of the approximate or abstracted system to the concrete system:

Definition 7. *Given transition systems* $T_1 = (S_1, I_1, R_1, V_1)$ *and* $T_2 = (S_2, I_2, R_2, V_2)$*, assume that* $T_1 \sqsubseteq_\alpha T_2$*, the ACTL formula* ϕ *describes properties of* T_2*, and* (α, γ) *forms a Galois connection.* $C(\phi)$ *represents a transformation on* ϕ *that descends on the subformulas recursively and transforms every atomic atomic formula* a *with* $\gamma(a)$ *(see [4] for details).*

For example, let ϕ be $AG(b_1 \vee b_2)$, where b_1 and b_2 represent $z = 1$ and $z < 1$, respectively, when the model in Figure 1 is predicate abstracted wrt to the set of predicates $\varphi = \{z = 1, z < 1\}$ and the Galois connection (α, γ) defined as in Equations 1 and 2. Then, $C(\phi) = AG(z \leq 1)$.

The preservation of ACTL properties when going from the approximate system to the concrete system is proved for existential abstraction in [4]. Here, we adapt it to an instantiation of existential abstraction using predicate abstraction as in [5]:

Lemma 3. *Assume* $T_1 \sqsubseteq_\alpha T_2$*,* ϕ *denotes an ACTL formula that describes a property of* T_2*,* $C(\phi)$ *denotes the transformation of the correctness property as in Definition 7, and* (α, γ) *forms a Galois connection and defines predicate abstraction and concretization as given in Equations 1 and 2, respectively. Then,* $T_2 \models \phi$ *implies* $T_1 \models C(\phi)$*.*

Proof. Preservation of atomic properties: *If a state* s_2 *in* T_2 *satisfies an atomic abstract property* ϕ*, due to the correctness preserving property of a Galois connection,* s_2 *also satisfies* $\gamma(\phi)$ *[14]. Due to soundness of the mapping between the states in* T_1 *to states in* T_2 *and monotonic property of* α *and* γ*, any state* s_1 *in* T_1 *that gets mapped to* s_2*, that is every state in* $\gamma(s_2)$ *also satisfies* $\gamma(\phi)$*.*
Preservation of ACTL Properties: *Follows from Corollary 1 in [4] and using* α *as the mapping function* h *in [4].*

3 A Hybrid Approach

In Section 3.1, we introduce a symbolic abstraction operator for transitions and an over-approximating abstract post operator derived from it. The abstract post operator enables partial predicate abstraction of an infinite-state system. Section 3.2 elaborates on the proposed hybrid approach that combines predicate abstraction and fixpoint approximations to perform CTL model checking of infinite-state systems. It also demonstrates soundness of the hybrid approach, which follows from the soundness results of the individual approaches and the over-approximating nature of the proposed abstract post operator.

3.1 Computing A Partially Predicate Abstracted Transition System

We compute an abstraction of a given transition system via a set of predicates such that only the variables that appear in the predicates disappear, i.e., existentially quantified, and all the other variables are preserved in their concrete domains and in the exact semantics from the original system. As an example,

using the set of predicates $\{z = 1, z < 1\}$, we can partially abstract the model in Figure 1 in a way that z is removed from the model, two new boolean variables b_1 (for $z = 1$) and b_2 (for $z < 1$) are introduced, and s, t, a_1, a_2, pc_1, and pc_2 remain the same as in the original model.

Abstracting transitions. A concrete transition $r^\natural$ is predicate abstracted using a mapping function α^τ via a set of current state predicates φ and a set of next state predicates φ' by introducing a predicate boolean variable b_i that represents predicate φ_i in the current state and a predicate boolean variable b'_i that represents predicate φ_i in the next state and existentially quantifying the current and next state concrete variables $V(\varphi) \cup V(\varphi')$ that appear in the current state and next state predicates:

$$\alpha^\tau(r^\natural) = \exists V(\varphi).\exists V(\varphi').(r^\natural \ \wedge \ CS \ \wedge \ \bigwedge_{i=1}^{|\varphi|} \varphi_i \iff b_i \wedge \bigwedge_{i=1}^{|\varphi|} \varphi'_i \iff b'_i), \quad (3)$$

where CS represents a consistency constraint that if all the abstracted variables that appear in a predicate remains the same in the next state then the corresponding boolean variable is kept the same in the next state:

$$CS = \bigwedge_{\varphi_i \in \varphi} ((\bigwedge_{v \in V(\varphi_i)} v' = v) \implies b'_i \iff b_i).$$

Concretization of abstract transitions. An abstract transition $r^\sharp$ is mapped back to all the concrete transitions it represents by replacing each current state boolean variable b_i with the corresponding current state predicate φ_i and each next state boolean variable b'_i with the corresponding next state predicate φ'_i:

$$\gamma^\tau(r^\sharp) = r^\sharp[\bar{\varphi}, \bar{\varphi}'/\bar{b}, \bar{b}']$$

For instance, for the model in Figure 1 and predicate set $\phi = \{z = 1, z < 1\}$, partial predicate abstraction of r_i^{cs}, $\alpha^\tau(r_i^{cs})$, is computed as

$$\begin{aligned} pc_i = & try \ \wedge \ s \geq a_i \ \wedge \ ((b_1 \wedge \neg b_2 \wedge \neg b'_1 \wedge \neg b'_2) \vee \ (\neg b_1 \wedge b_2 \wedge (b'_1 \vee b'_2)) \\ & \vee \ (\neg b_1 \wedge \neg b_2 \wedge \neg b'_1 \wedge \neg b'_2)) \ \wedge \ pc'_i = cs. \end{aligned} \quad (4)$$

It is important to note that the concrete semantics pertaining to the integer variables s and a_i and the enumerated variable pc_i are preserved in the partially abstract system.

Abstraction function α^τ represents a safe approximation for transitions:

Lemma 4. *$(\alpha^\tau, \gamma^\tau)$ defines a Galois connection (see the Appendix for the proof).*

One can compute an over-approximation to the set of reachable states via an over-approximating abstract post operator that computes the abstract successor states:

Lemma 5. *α^τ provides an over-approximate post operator:*

$$post[r^\natural](\gamma(s^\sharp)) \ \subseteq \ \gamma(post[\alpha^\tau(r^\natural)](s^\sharp))$$

Proof.

$$post[\tau^\natural](\gamma(s^\sharp)) \ \subseteq post[\gamma^\tau(\alpha^\tau(\tau^\natural))](\gamma(s^\sharp))(due\ to\ Lemma\ 4) \tag{5}$$

We need to show the following:

$$\begin{array}{rcl}
post[\gamma^\tau(\alpha^\tau(\tau^\natural))](\gamma(s^\sharp)) & \subseteq & \gamma(post[\alpha^\tau(\tau^\natural)](s^\sharp)) \\
post[\gamma^\tau(\tau^\sharp)](\gamma(s^\sharp)) & \subseteq & \gamma(post[\tau^\sharp](s^\sharp)) \\
(\exists V_\natural.\ \tau^\sharp[\bar{\varphi},\bar{\varphi}'/\bar{b},\bar{b}'] \ \wedge \ s^\sharp[\bar{\varphi}/\bar{b}])[V_\natural/V'_\natural] & \subseteq & (\exists V_\sharp.\ \tau^\sharp \ \wedge \ s^\sharp)[V_\sharp/V'_\sharp][\bar{\varphi}/\bar{b}] \\
(\exists V_\natural.\ \tau^\sharp[\bar{\varphi},\bar{\varphi}'/\bar{b},\bar{b}'] \ \wedge \ s^\sharp[\bar{\varphi}/\bar{b}])[V_\natural/V'_\natural] & \subseteq & (\exists V_\sharp.\ \tau^\sharp \ \wedge \ s^\sharp)[\bar{\varphi}'/\bar{b}'][V_\natural/V'_\natural] \\
(\exists V_\natural.(\ \tau^\sharp \ \wedge \ s^\sharp)[\bar{\varphi},\bar{\varphi}'/\bar{b},\bar{b}'])[V_\natural/V'_\natural] & \subseteq & (\exists V_\sharp.\ \tau^\sharp \ \wedge \ s^\sharp)[\bar{\varphi}'/\bar{b}'][V_\natural/V'_\natural]
\end{array} \tag{6}$$

$$post[\tau^\natural](\gamma(s^\sharp)) \ \subseteq \gamma(post[\alpha^\tau(\tau^\natural)](s^\sharp))(due\ to\ Equations\ 5\ \&\ 6) \tag{7}$$

3.2 Combining Predicate Abstraction with Fixpoint Approximations

At the heart of the hybrid approach is a partially predicate abstracted transition system and we are ready to provide a formal definition:

Definition 8. *Given a concrete infinite-state transition system $T^\natural = (S^\natural, I^\natural, R^\natural, V^\natural)$ and a set of predicates φ, where $V(\varphi) \subseteq V^\natural_{int}$, the partially predicate abstracted transition system $T^\sharp = (S^\sharp, I^\sharp, R^\sharp, V^\sharp)$ is defined as follows:*

- $S^\sharp \subseteq \mathcal{B}^{|V^\natural_{bool}|+|\varphi|} \times \mathcal{Z}^{|V^\natural_{int} \setminus V(\varphi)|}$
- $S^\sharp = \bigcup_{s^\natural \in S^\natural} \alpha(s^\natural)$.
- $I^\sharp = \bigcup_{is^\natural \in I^\natural} \alpha(is^\natural)$.
- $R^\sharp = \bigcup_{r^\natural \in R^\natural} \alpha^\tau(r^\natural)$.

A partially predicate abstracted transition system $T^\sharp$ defined via α and α^τ functions is a conservative approximation of the concrete transition system.

Lemma 6. *Let the abstract transition system $T^\sharp = (S^\sharp, I^\sharp, R^\sharp, V^\sharp)$ be defined as in Definition 8 with respect to the concrete transition system $T^\natural = (S^\natural, I^\natural, R^\natural, V^\natural)$ and the set of predicates φ. $T^\sharp$ approximates $T^\natural$: $T^\natural \sqsubseteq_\alpha T^\sharp$.*

Proof. It is straightforward to see, i.e., by construction, that $\exists s_1.(\alpha(s_1) = s_2 \ \wedge \ s_1 \in I^\natural)$ implies $s_2 \in I^\sharp$. To show $\exists s_1, s'_1.(\alpha(s_1) = s_2 \ \wedge \ \alpha(s'_1) = s'_2 \ \wedge \ (s_1, s'_1) \in R^\natural)$ implies $(s_2, s'_2) \in R^\sharp$, we need to show that $\exists s_1, s'_1.(\alpha(s_1) = s_2 \ \wedge \ \alpha(s'_1) = s'_2 \ \wedge \ s'_1 \in post[R^\natural](s_1))$ implies $s'_2 \in post[\alpha^\tau(R^\natural)](s_2)$, which follows from Lemma 5: $s'_1 \in \gamma(post[\alpha^\tau(R^\natural)](s_2))$ and $\alpha(s'_1) \in \alpha(\gamma(post[\alpha^\tau(R^\natural)](s_2)))$, and hence $s'_2 \in post[\alpha^\tau(R^\natural)](s_2)$.

Therefore, ACTL properties verified on $T^\sharp$ also holds for $T^\natural$:

Lemma 7. *Let the abstract transition system $T^\sharp = (S^\sharp, I^\sharp, R^\sharp, V^\sharp)$ be defined as in Definition 8 with respect to the concrete transition system $T^\natural = (S^\natural, I^\natural, R^\natural, V^\natural)$ and the set of predicates φ. Given an ACTL property $f^\sharp$, $T^\sharp \models f^\sharp \rightarrow T^\natural \models \gamma(f^\sharp)$.*

Proof. Follows from Lemmas 3 and 6.

Using fixpoint approximation techniques on an infinite-state partially predicate abstracted transition system in symbolic model checking of CTL properties [2] preserves the verified ACTL properties due to Lemma 1 and Lemma 7.

Restricting the state space of an abstract transition system $T^\sharp = (S^\sharp, I^\sharp, R^\sharp)$ with an over-approximation of the set of reachable states $T^\sharp_{RS} = (\mu Z.post[R^\sharp](Z) \vee I^\sharp)^+, I^\sharp, R^\sharp)$ also preserves the verified ACTL properties:

Lemma 8. *Let the abstract transition system $T^\sharp = (S^\sharp, I^\sharp, R^\sharp, V^\sharp)$ be defined as in Definition 8 with respect to the concrete transition system $T^\natural = (S^\natural, I^\natural, R^\natural, V^\natural)$. Let $T^\sharp_{RS} = ((\mu Z.I^\sharp \vee post[R^\sharp](Z))^+, I^\sharp, R^\sharp, V^\sharp)$. Given an ACTL property $f^\sharp$, $I^\sharp \subseteq [\![f^\sharp]\!]^-_{T^\sharp_{RS}} \rightarrow T^\natural \models \gamma(f^\sharp)$.*

Proof. Follows from Lemma 1 that approximate symbolic model checking is sound, i.e., $I^\sharp \subseteq [\![f^\sharp]\!]^-_{T^\sharp_{RS}}$ implies $T^\sharp \models f^\sharp$, and from Lemma 7 that ACTL properties verified on the partially predicate abstracted transition system holds for the concrete transition system, i.e., $T^\sharp \models f^\sharp$ implies $T^\natural \models \gamma(f^\sharp)$.

As an example, using the proposed hybrid approach one can show that the concrete model, $T^\natural_{ticket2}$ given in Figure 1 satisfies the correctness property $AG(z \leq 1)$ by first generating a partially predicate abstracted model, $T^\sharp_{ticket2}$, wrt the predicate set $\{z = 1, z < 1\}$ and performing approximate fixpoint computations to prove $AG(b_1 \vee b_2)$. Due to Lemma 8, if $T^\sharp_{ticket2,RS}$ satisfies $AG(b_1 \vee b_2)$, it can be concluded that $T^\natural_{ticket2}$ satisfies $AG(z \leq 1)$.

The main merit of the proposed approach is to combat the state explosion problem in the verification of problem instances for which predicate abstraction does not provide the necessary precision (even in the case of being embedded in a CEGAR loop) to achieve a conclusive result. In such cases approximate fixpoint computations may turn out to be more precise. The hybrid approach may provide both the necessary precision to achieve a conclusive result and an improved performance by predicate abstracting the variables that do not require fixpoint approximations. In Section 4, we present a crafted model that embeds bigger instances of the ticket mutual exclusion model and provide empirical evidence on the merits of the hybrid approach.

4 Experiments

The hypothesis we would like to test in this paper is that *the hybrid approach would be more effective than the individual techniques alone if the analysis does*

not converge without computing an approximation to the fixpoint and the number of integer variables pushes the limits of of the underlying symbolic, e.g., polyhedral, representation for the integer domain. The ticket mutual exclusion model shown in Figure 1 is too small to be a useful case study for the hypothesis that we set to test. So we combined a model for Airport Ground Network Traffic Control (AGNTC) [18] with the mutual exclusion algorithm shown in Figure 1. AGNTC is a resource sharing model for multiple processes, where the resources are taxiways and runways of an airport ground network and the processes are the arriving and departing airplanes. We changed the AGNTC model given in [18] by 1) using the mutual exclusion algorithm for synchronization on one of the taxiways and 2) making parked arriving airplanes fly and come back to faithfully include the mutual exclusion model, i.e., processes go back to *think* state after they are done with the critical section to attempt to enter the critical section again. The mutual exclusive use of the taxiway and the progress of an airplane attempting to use the taxiway could not be verified for the final model we obtained without using approximate fixpoint computations.

Although we will be using the AGNTC model to present our results, we would like to start by presenting verification results for the ticket mutual exclusion algorithm using ALV [18] and NuXmv [3]. ALV is a symbolic model checker that can represent the state space of an infinite-state system through a composition of symbolic representations so that each state variable can be encoded with the most suitable symbolic representation. It uses BDD representation for the boolean domain and supports two alternative representations for the integer domain: polyhedra based representation[2] (using the Omega library [12]) and automata based representation. ALV leverages the widening operator defined by the underlying integer representations to compute over-approximations for the fixpoint computations of EU, AU, EF, AF operators (-A flag) and the set of reachable states (-F flag). The reason we chose NuXmv is that it is a symbolic model checker that supports full CTL and implements an efficient CEGAR loop in connection with k-induction [16]. So we tried to verify safety property of the ticket model with both tools and the results are given in Figure 1. The experiments have been executed on a 64-bit Intel Xeon(R) CPU with 8 GB RAM running Ubuntu 14.04 LTS.

We varied the size of the model by varying the number of processes that try to have mutual exclusive access to the critical section. ALV using fixpoint approximations only could verify all 4 cases; the biggest model taking around 20 minutes, which is a perfect example of state-explosion as one smaller instance was verified in less than 4 secs. NuXmv's explicit predicate abstraction did not finish. So we tried implicit predicate abstraction. In that mode NuXmv could successfully verify the smallest instance in 2 secs by inferring the necessary predicates. However, for all the remaining instances it could neither prove nor falsify the models in 45 mins. So this experiment shows that the ticket mutual exclusion

[2] Experimental results are based on the widening operator implemented for the polyhedral representation.

algorithm is a benchmark for which widening based fixpoint approximation is more effective than predicate abstraction in a CEGAR loop.

Model	ALV (Appr.)		NuXmv (CEGAR, k-induction)	
	Memory	**Time**	**Memory**	**Time**
ticket2	1.73M	0.06	79.43 M	2.01 (bound=24, verified)
ticket3	3.71 M	0.88	-	>2700 (bound >115, unable to prove)
ticket4	6.64 M	3.49	-	>2700 (bound >58, unable to prove)
ticket5	280.77 M	1170.36	-	>2700 (bound >24, unable to prove)

Table 1. Comparison of ALV's fixpoint approximation based approach with NuXmv's CEGAR loop using predicate abstraction and k-induction.

To find out whether combining predicate abstraction with fixpoint approximations has any benefit, we extended ALV with predicate abstraction and conducted some experiments. In our implementation, we did not instantiate predicate abstraction in the context of a CEGAR loop. So we used predicates that can be easily inferred from the model. Therefore, for all the configurations we made sure that the set of predicates produce a conclusive result.

Table 2 shows sizes of the Airport Ground Network Control models for 2, 3, 4, and 5 arriving airplanes and one departing airplane. There are 2 taxiways and 2 runways. The number of integer variables, Int, and the number of boolean variables, $Bool$, are due to the state variables in the model for the fixpoint approximation only case. For the predicate abstraction, we used different sets of predicates for safety and liveness. In the case of safety, we abstracted the 6 integer variables that modeled the taxiways and runways and used 6 predicates whereas in the case of liveness, we abstracted two integer variables that represent the runways and used 2 predicates. The difference in the number of predicates are reflected in the memory consumption during transition system construction for safety verification and liveness verification cases. We used predicates in the form of $rc < 1$ and $rc = 1$ for verification of safety and liveness properties, respectively. Here, rc denotes the number of airplanes on ground network resource r.

	Fix. Apr.		**Pred. Abs + Fix. Apr.**			
	Safety & Liveness		Safety		Liveness	
	$M, \#I, \lvert BDD\rvert$	$Int, Bool$	$M, \#I, \lvert BDD\rvert$	$Int, Bool$	$M, \#I, \lvert BDD\rvert$	$Int, Bool$
2A	4.53, 354, 726	10, 10	214.66, 26, 566	4, 16	29.26, 102, 553	8, 12
3A	7.31, 547, 1590	11, 14	281.27, 43, 883	5, 20	45.79, 133, 922	9, 16
4A	10.32, 760, 2794	12, 18	348.08, 64, 1248	6, 24	64.33, 168, 1339	10, 20
5A	13.64, 993, 4338	13, 28	417.10, 89, 1661	7, 26	85.20, 207, 1804	11, 24

Table 2. Comparison of the transition system size in terms of memory (M) in MBs and the sizes of the symbolic representations (I for integer constraints, $|B|$ for BDDS size) and the number of integer (Int) and boolean variables ($Bool$) for fixpoint approximation only mode versus the mixed mode of predicate abstraction and fixpoint approximation.

Tables 3 and 4 show various statistics for safety and liveness verification of the Airport Ground Network Control model, respectively. *Size* column presents data about the last fixpoint iterate. Time includes the transition relation construction time and the verification time in seconds. *In the case of safety property verification, the combined approach provides significant improvements for the total time as the model gets bigger. The memory overhead could be tolerated as long as the memory is not exhausted. In the case of liveness property verification, the combined approach shows improvement both in terms of time and memory.* This is due to the greatest fixpoint computation (EG) for the negated property and the conjunction causing an exponential blow-up in the symbolic representation as the number of fixpoint iterations increase. For both safety and liveness verification, we used ALV's approximate reachable state computation.

Model	**Fix. Apr.**		**Pred. Abs + Fix. Apr.**	
	Size $M, \#Int, \|BDD\|$	**Time**	**Size** $M, \#Int, \|BDD\|$	**Time**
2A	10.02, 558, 791	1.76	414.67, 21, 1019	16.30
3A	32.46, 2176, 4466	59.29	511.78, 73, 3872	52.67
4A	95.76, 6072, 17586	1216.47	676.19 , 185, 10175	271.48
5A	245.92, 12272, 50162	11199.00	912.34, 381, 22850	639.33

Table 3. Comparison of using fixpoint approximation only versus a combination of predicate abstraction and fixpoint approximation for **safety property** verification of the Airport Ground Network Control model *using the approximate set of reachable states* (-F flags).

Model	**Fix. Apr.**		**Pred. Abs. + Fix. Apr.**	
	Size $M, \#Int, \|BDD\|$	**Time**	**Size** $M, \#Int, \|BDD\|$	**Time**
2A	47.84, 474, 680	5.71	43.76, 138, 412	3.85
3A	194.70, 1856, 3908	174.62	106.41, 538, 2101	96.81
4A	451.76, 5216, 15596	3142.21	273.78, 1625, 8006	1850.27
5A	>1772.30, 12272, 50139	> 11431.70	>288.60, 3980, 40434	>7867.00

Table 4. Comparison of using fixpoint approximation only versus a combination of predicate abstraction and fixpoint approximation for **liveness property** verification of the Airport Ground Network Control case study *using the approximate set of reachable states* (-F flag).

Unlike traditional implementation of predicate abstraction, our approach to generation of the abstract transition system does not involve weakest precondition computation with respect to the predicates. Also, our approach does not use an SMT solver, in the traditional sense, to compute the abstraction. This is because we literally perform the existential quantification in Equations 1 and 3 using ALV's polyhedral symbolic representation that uses the Omega Library [12]. However, in our approach the price is paid by having a blow-up in the abstract transition system that is exponential in the number of predicates. We deal

# of Preds.	2A			3A			4A		
	Reduction		Time	Reduction		Time	Reduction		Time
	$\#Int$	$\|BDD\|$		$\#Int$	$\|BDD\|$		$\#Int$	$\|BDD\|$	
1	%71.62	%59.68	0.29	%77.56	%66.56	0.53	% 80.66	%71.14	0.93
2	%85.47	%78.59	0.54	%88.35	%81.81	1.03	%89.87	%84.62	1.75
3	%93.27	%89.84	0.96	%94.64	%91.73	1.96	%95.35	%92.90	3.46
4	%97.34	%95.34	1.81	%97.78	%96.16	3.90	%98.01	%96.67	7.23
5	%98.82	%97.80	4.45	%98.99	%98.17	9.75	%99.11	%98.41	18.08
6	%99.56	%98.98	11.83	%99.60	%99.16	25.83	%99.62	%99.27	45.86

Table 5. Percentage of reductions achieved using ALV's simplification heuristic. The reductions are shown for the number of integer constraints ($\#Int$) and the sizes of the BDDs ($|BDD|$). The time taken by the simplification stage is given in secs.

with this blow-up by using the simplification heuristic of ALV [17] after the existential quantification is computed as part of the abstraction process (Equation 3). The existential quantification yields an exponentially large (in the number of predicates) transition system. Table 5 shows the reductions we obtained in the sizes of the transition relations and the time the simplification stages took. As the table shows the range of reductions is approximately [%60,%99.56] and the reductions increase with the increasing number of predicates as well as the model size.

Predicate Abstraction Only With ALV, generating a finite-state abstraction of the model using predicate abstraction only, i.e., abstracting all integer variables, has not been effective as it ended up in an out of memory error while building the abstract transition system. Since the predicate abstraction in ALV is not optimized, we wanted to use another tool that has an optimized implementation to see if using predicate abstraction alone would be more effective than the combined approach. As we did for the ticket example, we chose to use NuXmv. In the context of this model checking tool, explicit predicate abstraction did not scale as building the model for $2A$ instance did not finish in 40 minutes. NuXmv's implicit predicate abstraction, which can verify invariant properties of infinite-state systems using a combination of predicate abstraction inside a CEGAR loop and Bounded Model Checking, could neither verify nor refute the same model in 40 minutes. So even for the smallest instance of the case study, predicate abstraction alone has not been effective in verifying the safety property. Since NuXmv cannot handle liveness properties when it uses implicit predicate abstraction, we were not able to compare its performance for that case.

5 Related Work

In [10] a new abstract domain that combines predicate abstraction with numerical abstraction is presented. The idea is to improve precision of the analysis when the predicates involve numeric variables that are represented by an abstract numeric domain, e.g., a predicate on an array cell, where the domain of the index

variables are represented using polyhedral domain. In our approach the predicates and the numeric variables do not have any interference. Although [10] has evaluated the combined approach in the context of software model checking, our evaluation in the context of CTL model checking shows similar improvement in performance over complete numeric representation.

In [11] Jhala et al. point out the inadequacy of generating predicates for integer domain based on weakest preconditions over counter-examples. They propose a complete technique for finding effective predicates when the system satisfies the property and involves a bounded number of iterations. The technique limits the range of constants to be considered at each refinement stage and avoids generation of diverging predicates in the interpolation stage by discovering new constraints that relate program variables. Unlike the examples considered in [11], the presented mutual exclusion algorithm does not have a bound and, therefore, it is not obvious whether that technique would be successful on ticket-like models.

Transition predicate abstraction [15] is a technique that overcomes the inherent imprecision of state-based predicate abstraction with respect to proving liveness properties. Although we also consider primed versions of variables in the predicate as part of the transition, our approach cannot handle predicates that directly relate primed and unprimed variables. In [15] such predicates can be handled as it uses the abstract transitions to label nodes of the abstract program. However, our approach is able to handle liveness properties that relate abstracted and concrete variables.

6 Conclusion

We have implemented a hybrid approach that combines predicate abstraction with fixpoint approximations so that when approximate fixpoint computation is more effective than predicate abstraction in terms of providing the necessary precision, the state explosion can be dealt with the help of predicate abstraction. We have implemented the proposed approach in the context of Action Language Verifier, an infinite-state symbolic model checker that performs approximate CTL model checking. Experimental results show cases of improved performance for both safety and liveness verification when the hybrid approach is used. For future work, we would like to incorporate a CEGAR loop that would be capable of inferring a suitable partitioning of the state space between predicate abstraction and fixpoint approximations.

A Appendix

Let $\bar{A}$ denote an ordered list of terms from set A. Let $cube^{\bar{A}} = \bigwedge_{k=1}^{|A|} a_k$, where $a_k \in A$ or $\neg a_k \in A$ and k denotes the index of a_k in $\bar{A}$. Let $cube_i^{\bar{A}}$ denote the cube that evaluates to i when regarded as a $|A|$ bit number when the terms' encoding are interpreted as 0 for those that are negated and as 1 for the non-negated. For instance $cube_0^A$ denotes $\bigwedge_{k=1}^{|A|} \neg a_k$, where $a_k \in A$. Also, let $|\varphi| = n$.

Lemma 2.

Proof.

$$
\begin{aligned}
s \wedge cube_i^{\varphi} &\rightarrow \exists V_{\varphi}.s \wedge cube_i^{\varphi} \text{ (due to Existential Introduction)}\\
s \wedge cube_i^{\varphi} \wedge cube_i^{\varphi} &\rightarrow (\exists V_{\varphi}.s \wedge cube_i^{\varphi}) \wedge cube_i^{\varphi}\\
s \wedge cube_i^{\varphi} &\rightarrow (\exists V_{\varphi}.s \wedge cube_i^{\varphi}) \wedge cube_i^{\varphi}\\
\bigvee_{i=0}^{2^n-1} s \wedge cube_i^{\varphi} &\rightarrow \bigvee_{i=0}^{2^n-1} (\exists V_{\varphi}.s \wedge cube_i^{\varphi}) \wedge cube_i^{\varphi}\\
s \wedge \bigvee_{i=0}^{2^n-1} cube_i^{\varphi} &\rightarrow \bigvee_{i=0}^{2^n-1} (\exists V_{\varphi}.s \wedge cube_i^{\varphi}) \wedge cube_i^{\varphi}\\
s \wedge true &\rightarrow \bigvee_{i=0}^{2^n-1} (\exists V_{\varphi}.s \wedge cube_i^{\varphi}) \wedge cube_i^{\varphi}\\
s &\rightarrow \bigvee_{i=0}^{2^n-1} (\exists V_{\varphi}.s \wedge cube_i^{\varphi}) \wedge cube_i^{\varphi}\\
s &\rightarrow (\bigvee_{i=0}^{2^n-1} \exists V_{\varphi}.(s \wedge cube_i^{\varphi}) \wedge cube_i^{b}))[\bar{\varphi}/\bar{b}]\\
s &\rightarrow (\bigvee_{i=0}^{2^n-1} \exists V_{\varphi}.(s \wedge cube_i^{\varphi} \wedge cube_i^{b}))[\bar{\varphi}/\bar{b}] \text{ (due to } V_{\varphi} \cap V_b = \emptyset)\\
s &\rightarrow (\exists V_{\varphi}.(\bigvee_{i=0}^{2^n-1} s \wedge cube_i^{\varphi} \wedge cube_i^{b}))[\bar{\varphi}/\bar{b}]\\
s &\rightarrow (\exists V_{\varphi}.(s \wedge \bigvee_{i=0}^{2^n-1} cube_i^{\varphi} \wedge cube_i^{b}))[\bar{\varphi}/\bar{b}]\\
s &\rightarrow \gamma(\exists V_{\varphi}.(s \wedge \bigvee_{i=0}^{2^n-1} cube_i^{\varphi} \wedge cube_i^{b}))\\
s &\rightarrow \gamma(\exists V_{\varphi}.(s \wedge \bigwedge_{j=1}^{|\varphi|} \varphi_j \iff b_j))\\
s &\rightarrow \gamma(\alpha(s))
\end{aligned}
\tag{8}
$$

Lemma 4.

Proof. **Showing** $r^{\natural} \rightarrow \gamma^{\tau}(\alpha^{\tau}(r^{\natural}))$: Also, let $\overline{\varphi''}$ and $\overline{b''}$ denote the ordered list of terms from the set $\varphi \cup \varphi'$ and the ordered list of terms from the set $b \cup b'$, respectively.

Let $n = |\varphi| = |\varphi'|$ and $CS' = \bigwedge_{\varphi_i \in \varphi}((\bigwedge_{v \in V(\varphi_i)} v' = v) \implies \varphi'_i \iff \varphi_i)$.

$$
\begin{aligned}
r^\natural \wedge CS' \wedge cube_i^{\overline{\varphi''}} &\to \exists V_{\varphi\cup\varphi'}.r^\natural \wedge CS' \wedge cube_i^{\overline{\varphi''}} \\
r^\natural \wedge CS' \wedge cube_i^{\overline{\varphi''}} \wedge cube_i^{\overline{\varphi''}} &\to (\exists V_{\varphi\cup\varphi'}.r^\natural \wedge CS' \wedge cube_i^{\overline{\varphi''}}) \wedge cube_i^{\overline{\varphi''}} \\
r^\natural \wedge CS' \wedge cube_i^{\overline{\varphi''}} &\to (\exists V_{\varphi\cup\varphi'}.r^\natural \wedge CS' \wedge cube_i^{\overline{\varphi''}}) \wedge cube_i^{\overline{\varphi''}} \\
\bigvee_{i=0}^{2^{2n}-1} r^\natural \wedge CS' \wedge cube_i^{\overline{\varphi''}} &\to \bigvee_{i=0}^{2^{2n}-1} (\exists V_{\varphi\cup\varphi'}.r^\natural \wedge CS' \wedge cube_i^{\overline{\varphi''}}) \wedge cube_i^{\overline{\varphi''}} \\
r^\natural \wedge CS' \wedge \bigvee_{i=0}^{2^{2n}-1} cube_i^{\overline{\varphi''}} &\to \bigvee_{i=0}^{2^{2n}-1} (\exists V_{\varphi\cup\varphi'}.r^\natural \wedge CS' \wedge cube_i^{\overline{\varphi''}}) \wedge cube_i^{\overline{\varphi''}} \\
r^\natural \wedge true &\to \bigvee_{i=0}^{2^{2n}-1} (\exists V_{\varphi\cup\varphi'}.r^\natural \wedge CS' \wedge cube_i^{\overline{\varphi''}}) \wedge cube_i^{\overline{\varphi''}} \\
r^\natural &\to \bigvee_{i=0}^{2^{2n}-1} (\exists V_{\varphi\cup\varphi'}.r^\natural \wedge CS' \wedge cube_i^{\overline{\varphi''}}) \wedge cube_i^{\overline{\varphi''}} \\
r^\natural &\to (\bigvee_{i=0}^{2^{2n}-1} \exists V_{\varphi\cup\varphi'}.(r^\natural \wedge CS \wedge cube_i^{\overline{\varphi''}}) \wedge cube_i^{\overline{b''}}))[\overline{\varphi''}/\overline{b''}] \\
r^\natural &\to (\bigvee_{i=0}^{2^{2n}-1} \exists V_{\varphi\cup\varphi'}.(r^\natural \wedge CS \wedge cube_i^{\overline{\varphi''}} \wedge cube_i^{\overline{b''}}))[\overline{\overline{\varphi''}/\overline{b''}}] \\
&\quad \text{(due to } V_{\varphi\cup\varphi'} \cap V_b = \emptyset) \\
r^\natural &\to (\exists V_{\varphi\cup\varphi'}.(\bigvee_{i=0}^{2^{2n}-1} r^\natural \wedge CS \wedge cube_i^{\overline{\varphi''}} \wedge cube_i^{\overline{b''}}))[\overline{\varphi''}/\overline{b''}] \\
r^\natural &\to (\exists V_{\varphi\cup\varphi'}.(r^\natural \wedge CS \wedge \bigvee_{i=0}^{2^{2n}-1} cube_i^{\overline{\varphi''}} \wedge cube_i^{\overline{b''}}))[\overline{\varphi''}/\overline{b''}] \\
r^\natural &\to \gamma^\tau(\exists V_{\varphi\cup\varphi'}.(r^\natural \wedge CS \wedge \bigvee_{i=0}^{2^{2n}-1} cube_i^{\overline{\varphi''}} \wedge cube_i^{\overline{b''}})) \\
r^\natural &\to \gamma^\tau(\exists V_{\varphi\cup\varphi'}.(r^\natural \wedge CS \wedge \bigwedge_{j=1}^{|\varphi|} \varphi_j \iff b_j \wedge \bigwedge_{j=1}^{|\varphi'|} \varphi'_j \iff b'_j)) \\
r^\natural &\to \gamma^\tau(\alpha^\tau(r^\natural))
\end{aligned}
$$

Showing $r^\sharp = \alpha(\gamma(r^\sharp))$:

$$
\begin{aligned}
&\equiv \alpha(\gamma(r^\sharp)) \\
&\equiv \exists V(\varphi).\exists V(\varphi').(r^\sharp[\bar{\varphi}, \bar{\varphi}'/\bar{b}, \bar{b}'] \wedge CS \wedge \bigwedge_{i=1}^{|\varphi|} \varphi_i \iff b_i \wedge \bigwedge_{i=1}^{|\varphi|} \varphi'_i \iff b'_i) \\
&\equiv \exists V(\varphi).\exists V(\varphi').(r^\sharp[\bar{\varphi}, \bar{\varphi}'/\bar{b}, \bar{b}'] \wedge \bigwedge_{i=1}^{|\varphi|} \varphi_i \iff b_i \wedge \bigwedge_{i=1}^{|\varphi|} \varphi'_i \iff b'_i) \\
&\equiv r^\sharp
\end{aligned}
\tag{9}
$$

References

1. G. R. Andrews. *Concurrent Programming: Principles and Practice.* Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1991.
2. T. Bultan, R. Gerber, and W. Pugh. Symbolic model checking of infinite state systems using presburger arithmetic. In *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*, pages 400–411, 1997.
3. R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta. The nuxmv symbolic model checker. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, pages 334–342, 2014.
4. E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Trans. Program. Lang. Syst.*, 16(5):1512–1542, 1994.
5. E. M. Clarke, O. Grumberg, M. Talupur, and D. Wang. Making predicate abstraction efficient: How to eliminate redundant predicates. In *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, pages 126–140, 2003.
6. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.
7. P. Cousot and R. Cousot. On abstraction in software verification. In *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, pages 37–56, 2002.
8. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, Tucson, Arizona, USA, January 1978*, pages 84–96, 1978.
9. S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*, pages 72–83, 1997.
10. A. Gurfinkel and S. Chaki. Combining predicate and numeric abstraction for software model checking. *STTT*, 12(6):409–427, 2010.
11. R. Jhala and K. L. McMillan. A practical and complete approach to predicate refinement. In *Proceedings of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'06, pages 459–473, Berlin, Heidelberg, 2006. Springer-Verlag.
12. W. Kelly, V. Maslov, W. Pugh, E. Rosser, T. Shpeisman, and D. Wonnacott. The omega library interface guide. Technical report, University of Maryland at College Park, College Park, MD, USA, 1995.
13. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Form. Methods Syst. Des.*, 6(1):11–44, Jan. 1995.
14. F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.
15. A. Podelski and A. Rybalchenko. Transition predicate abstraction and fair termination. *ACM Trans. Program. Lang. Syst.*, 29(3), May 2007.

16. S. Tonetta. Abstract model checking without computing the abstraction. In A. Cavalcanti and D. Dams, editors, *FM 2009: Formal Methods*, volume 5850 of *Lecture Notes in Computer Science*, pages 89–105. Springer Berlin Heidelberg, 2009.
17. T. Yavuz-Kahveci and T. Bultan. Heuristics for efficient manipulation of composite constraints. In *Frontiers of Combining Systems, 4th International Workshop, FroCoS 2002, Santa Margherita Ligure, Italy, April 8-10, 2002, Proceedings*, pages 57–71, 2002.
18. T. Yavuz-Kahveci and T. Bultan. Action Language verifier: an infinite-state model checker for reactive software specifications. *Formal Methods in System Design*, 35(3):325–367, 2009.