Abstract Model Checking without Computing the Abstraction

Stefano Tonetta[⋆]

FBK-Irst
tonettas@fbk.eu

Abstract. Abstraction is a fundamental technique that enables the verification of large systems. In symbolic model checking, abstractions are defined by formulas that relate concrete and abstract variables. In predicate abstraction, the abstract variables are equivalent to some predicates over the concrete variables.

In order to apply model checking on the abstract state space, it is usually necessary to compute a quantifier-free formula that is equivalent to the abstract transition relation. In predicate abstraction, the quantifier elimination can be obtained by solving an ALLSAT problem. In many practical cases, this computation results into a bottleneck.

In this paper, we propose a new algorithm that combines abstraction with bounded model checking and k-induction. The algorithm does not rely on quantifier elimination, but encodes the model checking problem over the abstract state space into SAT problems. The algorithm is a novelty in the state-of-the-art of abstract model checking because it avoids computing the abstraction. An experimental evaluation with case studies taken from an industrial project shows that the new algorithm is more efficient and reaches in some cases a time improvement that is exponential in the number of predicates.

1 Introduction

Model Checking (MC) [14,26] is an automatic technique to verify if a system satisfies a property. The main problem of MC is the state-space explosion, i.e., the system is often too large to be verified. *Abstraction* [8] and symbolic representation [3] are two major classes of techniques broadly adopted to tackle the problem.

Abstraction defines a relationship between the states of the concrete system and the states of a smaller system, which is more amenable for the verification. Typically, abstraction over-approximates the semantics of the system so that if a property holds in the abstract system, then it holds also in the concrete one. A particular abstraction technique widely used with MC is *predicate abstraction*

[⋆] The author would like to thank the Provincia Autonoma di Trento for the support with the project ANACONDA, and A. Cimatti, M. Roveri, and V. Schuppan for helpful discussions and feedback.

[15,11], where a set of predicates is chosen so that the abstract state space observes only the evolution of these predicates in the original system.

Symbolic model checking [3] represents sets of states and transitions with formulas. Symbolic algorithms manipulate formulas exploiting BDD-based or SAT-based techniques. SMT solvers, which combine SAT techniques with solvers for decidable first-order theories, are used when the system is described with first-order formulas. Bounded Model Checking (BMC) [2] is a symbolic technique that looks for counterexamples only with a bounded length k. BMC problems are encoded into SAT problems, which are then solved using either a SAT solver or an SMT solver, depending on the language used to describe the system. k-induction [27] is a technique used to prove that, if there is no counterexample of length up to k, then we can conclude that the property holds. The conditions to prove that k is sufficiently large are encoded into SAT problems.

In order to combine symbolic model checking with abstraction, it is necessary to compute a quantifier-free formula representing the transition relation of the abstract system. This is typically obtained by eliminating the quantifiers in the definition of the abstraction. When the abstract state space is finite, as in the case of predicate abstraction, the abstract transition relation can be obtained by solving an ALLSAT problem, i.e., by enumerating the models satisfying the formula that defines the abstraction. In some cases, however, quantifier elimination is not possible, and in practice, also for predicate abstraction, results to be a bottleneck (see, e.g., [10,19,4]).

In this paper, we propose a new algorithm for solving the problem of model checking an abstract system. The idea is to embed the definition of the abstraction in the BMC and k-induction encodings. This way, we can verify the correctness of the abstract system without computing the abstraction. The algorithm can be applied to predicate abstraction, but also to infinite-state abstraction such as abstraction obtained by projection. We extend the abstract BMC and k-induction to check the language emptiness of infinite-state fair transition systems.

With regard to the standard approach to abstract model checking which computes the abstraction upfront, the new algorithm considers only the parts of the abstract state space that are relevant to the search. The solver is used to solve a satisfiability problem rather than to enumerate all possible solutions. With regard to model checking the concrete state space, the new algorithm exploits the abstraction and solves more problems. When the abstract state space is finite as in the case of predicate abstraction, k-induction is guaranteed to terminate, which is not the case for the concrete infinite-state space.

We performed an experimental evaluation on benchmarks extracted from an industrial project on requirements validation. The results show that the new algorithm can solve problems where the computation of the abstraction is not feasible, while in general is more efficient and can yield a time improvement that is exponential in the number of predicates.

The paper is structured as follows: in Sec. 2, we describe the project that motivated our research; in Sec. 3, we overview the background of symbolic model checking and abstraction; in Sec. 4, we present the new algorithm for model checking an abstract system; in Sec. 5, we present the experimental evaluation; in Sec. 6, we discuss the related work; finally, in Sec. 7, we conclude and hint some future directions.

2 Motivations

2.1 Requirements Validation

Our work is motivated by a project funded by the European Railway Agency (`http://www.era.europa.eu`). The aim of the project was to develop a methodology supported by a tool for the validation of the System Requirement Specification of the European Train Control System (ETCS). The ETCS specification is a set of requirements related to the automatic supervision of the location and speed performed by the train on-board system. Building on emerging formal methods for requirements validation, the requirements are formalized in first-order temporal formulas and the analysis is based on a series of satisfiability problems [5]. The satisfiability of the formulas is reduced to the problem of language emptiness of fair transition systems. We use a BMC encoding with loops to find accepting paths in the transition systems. When the problem cannot be solved with BMC, we use predicate abstraction to check if the language of the transition system is empty.

The systems generated by the process of requirements validation are very large including hundreds of Boolean variables and tens of real variables. Unlike in the standard setting of MC, there is no property that can guide the abstraction and the source of inconsistency is not known a priori. Even when a set of predicates of the abstraction is manually defined, proving the language emptiness of such systems is challenging, because the computation of the abstraction becomes prohibitive with a dozen of predicates.

3 Background

3.1 Fair Transition Systems

Fair Transition Systems (FTSs) [21] are a symbolic representation of infinite-state automata. In symbolic model checking [3], FTSs are used to represent both the system and the property. Set of states are expressed by means of logical formulas over a given set $\mathcal{V}$ of variables, while set of transitions are represented by formulas over $\mathcal{V}$ and the set $\mathcal{V}'$ of next variables $\{v'\}_{v\in\mathcal{V}}$, where v' represents the next value of v. Symbolic algorithms manipulate such formulas in order to check if the system satisfies the property. We assume that the formulas belong to a decidable fragment of first-order logic for which we have a satisfiability solver. We use the abbreviations sat and unsat for satisfiable and unsatisfiable, resp.

Definition 1. *A Fair Transition System (FTS) is a tuple* $\langle \mathcal{V}, I, T, \mathcal{F} \rangle$, *where*

- $\mathcal{V}$ *is the set of variables,*
- $I(\mathcal{V})$ *is a formula that represents the initial condition,*
- $T(\mathcal{V}, \mathcal{V}')$ *is a formula that represents the transition relation,*
- $\mathcal{F} = \{F_1, ..., F_n\}$ *is a set of formulas representing the fairness conditions.*

The set $\mathcal{S}_{\mathcal{V}}$ of states is given by all truth assignments to the variables $\mathcal{V}$. Given a state s, we use s' to denote the corresponding truth assignment to the next state variables, i.e. $s' = s[\mathcal{V}'/\mathcal{V}]$. A state s is initial iff $s \models I(\mathcal{V})$. Given two states s_1 and s_2, there exists a transition between s_1 and s_2 iff $s_1, s_2' \models T(\mathcal{V}, \mathcal{V}')$. If π is a sequence of states we denote with π_i the i-th state of the sequence. If π is finite we denote with $|\pi|$ the length of π. A finite [resp. infinite] path of an FTS is a finite [resp. infinite] sequence of states π such that, for all i, $1 \leq i < |\pi|$ [resp. $i \geq 1$], there exists a transition between π_i and π_{i+1} ($\pi_i, \pi'_{i+1} \models T$). A path π is initial iff π_1 is an initial state ($\pi_1 \models I$). An infinite path π is fair iff, for all $F \in \mathcal{F}$, for infinitely many i, $\pi_i \models F$.

Given an FTS M and a formula $\varphi(\mathcal{V})$, the reachability problem is the problem of finding an initial finite path $s_0, ..., s_k$ of M such that $s_k \models \varphi$. Model checking of invariant properties can be reduced to a reachability problem.

The language of an FTS is given by the set of all initial fair paths. Given an FTS M, the language emptiness problem is the problem of finding an initial fair path of M. Model checking and satisfiability of linear-time temporal formulas are typically reduced to the emptiness language problem of equivalent FTSs [28]. Thus, also the validation of requirements expressed in temporal logic is typically solved by looking for an initial fair path in the FTS equivalent to. the conjunction of the requirements.

Example 1. Consider a system $\langle \{\delta_t, t, e\}, \mathit{true}, T_c \wedge T_d \wedge T_e, \{F_d, F_e\} \rangle$ where:

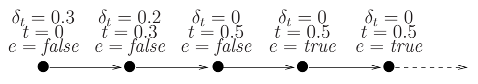
- δ_t is a non-negative real variable representing the time elapsed at every step.
- t is a timer (real variable) that progresses when $\delta_t > 0$ and can be reset when $\delta_t = 0$.
- e is a Boolean variable that can change only when $\delta_t = 0$.
- $T_c := \delta_t > 0 \rightarrow (e' = e \wedge t' - t = \delta_t)$ represents the constraint of a step with elapsing time.
- $T_d := \delta_t = 0 \rightarrow (t' = t \vee t' = 0)$ represents the constraint of a discrete step.
- $T_e := e \rightarrow (e' \wedge t' = t)$ states that when e becomes (non-deterministically) true both e and t do not change anymore.
- $F_d := \delta_t > 0$ forces the progress of time.
- $F_e := e$ forces the Boolean variable e to become eventually true.

The system has an infinite number of initial paths that reach the formula e, but no infinite fair path, because when e becomes true the time is forced to freeze. For example, the path shown in Fig. 1 is an initial path of the system, but cannot be extended to any fair path.

$S_{\mathcal{V}} \to S_{\hat{\mathcal{V}}}$ $\alpha(s) = \hat{s}$

$\gamma(\hat{s}) = \{ s \in S_{\mathcal{V}} \mid \alpha(s) = \hat{s} \}$

concretization abstraction function



$\delta_t = 0.3$ $t = 0$ $e = false$
$\delta_t = 0.2$ $t = 0.3$ $e = false$
$\delta_t = 0$ $t = 0.5$ $e = false$
$\delta_t = 0$ $t = 0.5$ $e = true$
$\delta_t = 0$ $t = 0.5$ $e = true$

Fig. 1. Example of path

$H_\alpha(\mathcal{V}, \hat{\mathcal{V}})$

3.2 Abstraction

Abstraction [8] is used to reduce the search space while preserving the satisfaction of some properties. In MC, the abstraction is usually obtained by means of a surjective function $\alpha : \mathcal{S}_{\mathcal{V}} \to \mathcal{S}_{\hat{\mathcal{V}}}$, called abstraction function, that maps states of an FTS M into states of a smaller FTS $\hat{M}$. The concretization function $\gamma : \mathcal{S}_{\hat{\mathcal{V}}} \to 2^{\mathcal{S}_{\mathcal{V}}}$ is defined as $\gamma(\hat{s}) = \{s \in \mathcal{S}_{\mathcal{V}} \mid \alpha(s) = \hat{s}\}$. The abstraction function α is symbolically represented by a formula $H_\alpha(\mathcal{V}, \hat{\mathcal{V}})$ such that $s, \hat{s} \models H_\alpha$ iff $\alpha(s) = \hat{s}$.

Once the set $\hat{\mathcal{V}}$ of abstract variables and the relation H_α are given, the abstraction $\hat{M}_\alpha$ is obtained by existentially quantifying the variables of M.

Definition 2. *Given an FTS M, a set $\hat{\mathcal{V}}$ of abstract variables and the relation H_α, the abstract FTS $\hat{M}_\alpha = \langle \hat{\mathcal{V}}, \hat{I}_\alpha, \hat{T}_\alpha, \hat{\mathcal{F}}_\alpha \rangle$ is defined as follows:*

- $\hat{I}_\alpha(\hat{\mathcal{V}}) := \exists \mathcal{V}(I(\mathcal{V}) \wedge H_\alpha(\mathcal{V}, \hat{\mathcal{V}}))$,
- $\hat{T}_\alpha(\hat{\mathcal{V}}, \hat{\mathcal{V}}') := \exists \mathcal{V} \exists \mathcal{V}'(T(\mathcal{V}, \mathcal{V}') \wedge H_\alpha(\mathcal{V}, \hat{\mathcal{V}}) \wedge H_\alpha(\mathcal{V}', \hat{\mathcal{V}}'))$.
- $\hat{\mathcal{F}}_\alpha = \{\hat{F}_\alpha\}_{F \in \mathcal{F}}$ *where* $\hat{F}_\alpha(\hat{\mathcal{V}}) := \exists \mathcal{V}(F(\mathcal{V}) \wedge H_\alpha(\mathcal{V}, \hat{\mathcal{V}}))$.

$\hat{I}_\alpha(\hat{\mathcal{V}}) := \exists \mathcal{V}(I(\mathcal{V}) \wedge H_\alpha(\mathcal{V}, \hat{\mathcal{V}}))$

$T_\alpha(\hat{\mathcal{V}}, \hat{\mathcal{V}}') = \exists \mathcal{V} \exists \mathcal{V}'(T(\mathcal{V}, \mathcal{V}') \wedge H_\alpha(\mathcal{V}, \hat{\mathcal{V}})$
$\wedge H_\alpha(\mathcal{V}', \hat{\mathcal{V}}')$

Given a formula φ, we define its abstract version $\hat{\varphi}_\alpha$ as $\exists \mathcal{V}(\varphi(\mathcal{V}) \wedge H_\alpha(\mathcal{V}, \hat{\mathcal{V}}))$.

The abstraction over-approximates the reachability of an FTS M, in the sense that if a condition is reachable in M, then also its abstract version is reachable in $\hat{M}_\alpha$. Similarly, if M has an initial fair path, the same holds for $\hat{M}_\alpha$. Thus, if we prove that a set of states is not reachable in $\hat{M}_\alpha$, or that $\hat{M}_\alpha$ does not have any initial fair path, the same can be concluded for the concrete FTS M.

Predicate abstraction. In Predicate Abstraction [15], the abstract state-space is described with a set of predicates; each predicate is represented by an abstract variable. Given an FTS M, we select a set $\mathbb{P}$ of predicates, such that each predicate $P \in \mathbb{P}$ is a formula over the variables $\mathcal{V}$ that characterizes relevant facts of the system. For every $P \in \mathbb{P}$, we introduce a new abstract variable v_P and define $\mathcal{V}_{\mathbb{P}}$ as $\{v_P\}_{P \in \mathbb{P}}$.

The abstraction function $\alpha_{\mathbb{P}}$ is defined as $\alpha_{\mathbb{P}}(s) := \{v_P \in \mathcal{V}_{\mathbb{P}} \mid s \models P\}$, while $H_{\mathbb{P}}$ is defined as follows:

$$H_{\mathbb{P}}(\mathcal{V}, \mathcal{V}_{\mathbb{P}}) := \bigwedge_{P \in \mathbb{P}} v_P \leftrightarrow P(\mathcal{V}) \tag{1}$$

Example 2. Consider the system of Ex. 1 and the predicates $\delta_t > 0$ and e. If we eliminate the quantifiers in the definition of abstraction, we obtain the FTS $\langle \{v_{\delta_t}, v_e\}, true, v_e \to (v_e \wedge \neg v_{\delta_t}), \{v_{\delta_t}, v_e\} \rangle$. This abstract system has finite states and it is easy to see that it does not have fair paths.

3.3 Bounded Model Checking

Bounded Model Checking (BMC) ([2]) considers only initial paths of length up to a certain bound. Given a bound k, an FTS M and a formula $\varphi(\mathcal{V})$, the bounded reachability problem is the problem of finding, for some $j \leq k$ an initial finite path $s_0, ..., s_j$ of M such that $s_j \models \varphi$. The problem is usually reduced to a satisfiability problem.

In the following, given a set X of variables, we use several copies of X, one for each state of the path we are looking for. We will denote with X_i the i-th copy of X. Thus, $V_i = \{v_i\}_{v \in \mathcal{V}}$, where v_i represents the i-th copy of v.

Definition 3. *Given an FTS M, and a bound k, the formula $PATH_{M,k}$ is defined as follows:*

$$PATH_{M,k} := \bigwedge_{1 \leq h \leq k} T(\mathcal{V}_{h-1}, \mathcal{V}_h) \tag{2}$$

Definition 4. *Given an FTS M, a bound k, and a formula φ, the formula $BMC_{M,k,\varphi}$ is defined as follows:*

$$BMC_{M,k,\varphi} := I(\mathcal{V}_0) \wedge PATH_{M,k} \wedge \varphi(\mathcal{V}_k) \tag{3}$$

The formula $BMC_{M,k,\varphi}$ encodes the bounded reachability problem.

Theorem 1. *$BMC_{M,k,\varphi}$ is sat iff there exists an initial path of length k reaching φ.*

Bounded model checking with fair paths. A similar encoding is often used to find lasso-shape initial fair paths of length up to k.

Definition 5. *Given an FTS M, and a bound k, the formula $BMC^{loop}_{M,k}$ is defined as follows:*

$$BMC^{loop}_{M,k} := I(\mathcal{V}_0) \wedge PATH_{M,k} \wedge \bigvee_{0 \leq l < k,\, v \in V} \bigwedge v_l = v_k \wedge \bigwedge_{F \in \mathcal{F},\, l \leq h < k} \bigvee F(\mathcal{V}_h) \tag{4}$$

As $BMC_{M,k,\varphi}$ is an approximation of the reachability problem, $BMC^{loop}_{M,k}$ is an approximation of the language emptiness problem: if the formula is sat, the language is not empty; otherwise, we have to increase the bound. However, if the state space is not finite, it is not guaranteed that there exists a bound sufficient to solve the problem.

3.4 K-Induction

K-induction [27] is a technique that proves that if a set of states is not reachable in k steps, then it is not reachable at all. On the lines of the induction principle, it consists of a base step, which solves the bounded reachability problem with a given bound k of steps, and an inductive step, which concludes that k is sufficient

```
input  : FTS $M = \langle \mathcal{V}, I, T \rangle$ and formula $\varphi$
output : $YES$ if $\varphi$ is reachable in $M$, $NO$ otherwise
1 begin
2     $k$:=0;
3     if $BMC_{M,k,\varphi}$ is sat then
4         return $YES$
5     else if $KINDFW_{M,k+1}$ or $KINDBW_{M,k+1,\varphi}$ is unsat then
6         return $NO$
7     else
8         k++;
9 end
```

Algorithm 1. K-induction(KIND)

to solve the (unbounded) reachability problem. The idea of the inductive step is to check either if the initial states cannot reach new (non-visited) states in $k+1$ steps, or the target set of states cannot be reached in $k+1$ steps. These checks can be solved by means of satisfiability.

$\neg a \wedge \neg b \wedge \neg c$

$\neg (a \vee b \vee c)$

Definition 6. *Given an FTS M, and a bound k, the formula $SIMPLEPATH_{M,k}$ is defined as follows:*

$$SIMPLEPATH_{M,k} := PATH_{M,k} \wedge \bigwedge_{0 \le i < j \le k} \neg \bigwedge_{v \in \mathcal{V}} v_i = v_j \tag{5}$$

$\neg(v_0 = v_1) \wedge \neg(v_0 = v_2)$

$\wedge \cdot \cdot \neg(v_0 = v_k)$

$\wedge \neg(v_1 = v_2) \wedge \neg(v_1 = v_3)$

$\cdot \cdot \cdot \wedge \neg(v_1 = v_k)$

$\wedge \neg(v_{k-1} = v_k)$

$\rightarrow \big(\underline{v_0 = v_1 \vee v_0 = v_2} \cdots$

$\vee \underline{v_0 = v_k \vee v_1 = v_3} \cdots$

$\vee \underline{v_1 = v_k} \cdots$

$\vee \underline{v_{k-1} = v_k} \big)$

Definition 7. *Given an FTS M, and a bound k, the formula $KINDFW_{M,k}$ is defined as follows:*

$$KINDFW_{M,k} := I(\mathcal{V}_0) \wedge SIMPLEPATH_{M,k} \tag{6}$$

Theorem 2. *If $KINDFW_{M,k+1}$ is unsat, then M does not have an initial simple path with more than k states.*

Definition 8. *Given an FTS M, a formula φ, and a bound k, the formula $KINDBW_{M,k,\varphi}$ is defined as follows:*

$$KINDBW_{M,k,\varphi} := SIMPLEPATH_{M,k} \wedge \varphi(\mathcal{V}_k) \tag{7}$$

Theorem 3. *If $KINDBW_{M,k+1,\varphi}$ is unsat, M does not have a simple path reaching φ with more than k states.*

Corollary 1. *If, for all $i \le k$, $BMC_{M,i,\varphi}$ is unsat and, either $KINDFW_{M,k+1}$ or $KINDBW_{M,k+1,\varphi}$ is unsat as well, then φ is not reachable in M.*

Corollary 1 gives rise to Algorithm 3.4, where the formulas are iteratively checked for increasing values of k. In case M is finite-state, Algorithm 3.4 is guaranteed to terminate. The works in [27] and [13] exploit stronger version of $KINDFW_{M,k}$ and $KINDBW_{M,k,\varphi}$ that consider the negation of the initial condition I and the target condition φ respectively.

$i = \{0, 1, 2\}$ $j = \{1, 2\}$

$i = 0$ $j = 1$ $\neg(x_0 = x_1) \vee \neg(y_0 = y_1)$

$j = 2$ $\neg(x_0 = x_2) \vee \neg(y_0 = y_2)$

$i = 1$, $j = 2$ $\neg(x_1 = x_2) \vee \neg(y_1 = y_2)$

$i = 2$ $\neg(\underline{x_0 = x_1 \wedge y_0 = y_1}) \wedge \neg(x_0 = x_2 \wedge y_0 = y_2)$

$\boxed{v_0 = v_1}$

$\neg$

$\neg(\boxed{\neg v_0 = v_1} \vee$

$i = 0$, $j = 1$

$v : \{x, y\}$

$\neg(x_0 = x_1) \vee \neg(y_0 = y_1)$

$\neg(x_0 = x_1) \vee \neg(y_0 = y_1)$

$\neg(x_1 = x_2) \vee \neg(x_1 = x_2)$

3.5 Abstract Model Checking with Abstraction

The standard way to solve the reachability problem or the emptiness problem on the abstraction of an FTS M is first to compute the FTS $\hat{M}$ and then to apply model checking techniques on the abstract state space. We denote with AMC$_{reach}$ such procedure when it solves the reachability problem, while with AMC$_{loop}$ the similar procedure that checks the language emptiness of the abstraction of M.

4 Abstract Model Checking without Abstraction

4.1 General Idea

The key idea of the paper is to embed the definition of the abstraction in the encoding of BMC. This highlights the possibility of pre-computing the quantification of the abstract variables. Let us consider the abstract version of the formula 3, namely:

$$\hat{I}_\alpha(\hat{\mathcal{V}}_0) \wedge \bigwedge_{1\leq h\leq k} \hat{T}_\alpha(\hat{\mathcal{V}}_{h-1}, \hat{\mathcal{V}}_h) \wedge \hat{\varphi}_\alpha(\hat{\mathcal{V}}_k) \equiv$$
$$\hat{I}_\alpha(\hat{\mathcal{V}}_0) \wedge \hat{T}_\alpha(\hat{\mathcal{V}}_0, \hat{\mathcal{V}}_1) \ldots \wedge \hat{T}_\alpha(\hat{\mathcal{V}}_{k-1}, \hat{\mathcal{V}}_k) \wedge \hat{\varphi}_\alpha(\hat{\mathcal{V}}_k)$$

If we substitute $\hat{I}_\alpha$, $\hat{T}_\alpha$, and $\hat{\varphi}_\alpha$ with their definitions, we obtain:

$$I(\mathcal{V}_0) \wedge H_\alpha(\mathcal{V}_0, \hat{\mathcal{V}}_0) \wedge H_\alpha(\overline{\mathcal{V}}_0, \hat{\mathcal{V}}_0) \wedge T(\overline{\mathcal{V}}_0, \mathcal{V}_1) \wedge H_\alpha(\mathcal{V}_1, \hat{\mathcal{V}}_1) \wedge \ldots \wedge$$
$$H_\alpha(\overline{\mathcal{V}}_{k-1}, \hat{\mathcal{V}}_{k-1}) \wedge T(\overline{\mathcal{V}}_{k-1}, \mathcal{V}_k) \wedge H_\alpha(\mathcal{V}_k, \hat{\mathcal{V}}_k) \wedge H_\alpha(\overline{\mathcal{V}}_k, \hat{\mathcal{V}}_k) \wedge \varphi(\overline{\mathcal{V}}_k)$$

where quantifiers have been lifted at top level by renaming some bound variables.

Note that the scope of abstract variables $\hat{\mathcal{V}}_i$ is limited to two copies of the abstraction relation. Let us define the formula $EQ_\alpha(\mathcal{V}, \overline{\mathcal{V}})$ as

$$EQ_\alpha(\mathcal{V}, \overline{\mathcal{V}}) := \exists\hat{\mathcal{V}}(H_\alpha(\mathcal{V}, \hat{\mathcal{V}}) \wedge H_\alpha(\overline{\mathcal{V}}, \hat{\mathcal{V}})) \quad (8)$$

EQ_α encodes the fact that two concrete states correspond to the same abstract state. Formally, $s, \overline{s} \models EQ_\alpha$ iff $\alpha(s) = \alpha(\overline{s})$. We can use EQ_α to provide abstract versions of the formulas used for BMC and k-induction. Intuitively, instead of having a contiguous sequence of transitions, the encoding represents a sequence of disconnected transitions where every gap between two transitions is forced to lay in the same abstract state (see Fig. 2).

In most of abstraction, the quantifier in EQ_α can be easily eliminated. For example in predicate abstraction:

$$EQ_\alpha \equiv EQ_{\mathbb{P}}(\mathcal{V}, \overline{\mathcal{V}}) := \bigwedge_{P\in\mathbb{P}} P(\mathcal{V}) \leftrightarrow P(\overline{\mathcal{V}}) \quad (9)$$

Another interesting case is the abstraction by projection where the abstract variables are a subset $\tilde{\mathcal{V}}$ of the concrete variables and the non-abstract variables are quantified out. In this case,

$$EQ_\alpha \equiv EQ_{\tilde{\mathcal{V}}}(\mathcal{V}, \overline{\mathcal{V}}) := \bigwedge_{v\in\tilde{\mathcal{V}}} v = \overline{v} \quad (10)$$

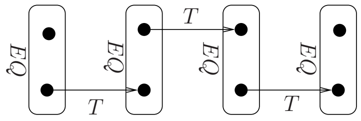


Fig. 2. Abstract path

4.2 Paths and Simple Paths

We first define the abstract version of the $PATH_{M,k}$ and $SIMPLEPATH_{M,k}$ used in the encoding of BMC and k-induction.

Definition 9. *Given an FTS $M = \langle \mathcal{V}, I, T \rangle$, an abstraction function α, and a bound k, the formula $\widehat{PATH}_{M,\alpha,k}$ is defined as follows:*

$\neg a \vee \neg b$

$\neg(\underline{a \wedge b})$ $\neg(\underline{a \vee b \vee \ldots})$

$$\widehat{PATH}_{M,\alpha,k} := \bigwedge_{1 \le h < k} (T(\mathcal{V}_{h-1}, \overline{\mathcal{V}}_h) \wedge EQ_\alpha(\overline{\mathcal{V}}_h, \mathcal{V}_h)) \wedge T(\mathcal{V}_{k-1}, \mathcal{V}_k) \qquad (11)$$

[illegible] [illegible]

$\neg(x_0 = x_1) \vee \neg(y_0 = y_1)$ $\neg a \wedge \neg b \wedge \neg c$

$\neg(x_0 = x_1 \wedge y_0 = y_1)$

$\neg(\mathcal{V}_0 = \mathcal{V}_1) \wedge \neg(\mathcal{V}_0 = \mathcal{V}_2) \wedge \neg(\mathcal{V}_0 = \mathcal{V}_3)$

$\wedge \cdots \neg(\mathcal{V}_{n-1} = \mathcal{V}_n)$

Theorem 4. *$\widehat{PATH}_{M,\alpha,k}$ is sat iff $PATH_{\hat{M}_\alpha,k}$ is sat.*

Definition 10. *Given an FTS $M = \langle \mathcal{V}, I, T \rangle$, an abstraction function α, and a bound k, the formula $\widehat{SIMPLEPATH}_{M,\alpha,k}$ is defined as follows:*

$EQ_\alpha(\mathcal{V}_i, \mathcal{V}_j) \triangleq \exists \hat{\mathcal{V}} (H_\alpha(\mathcal{V}_i, \hat{\mathcal{V}})$
$\wedge H_\alpha(\mathcal{V}_j, \hat{\mathcal{V}})$

$$\widehat{SIMPLEPATH}_{M,\alpha,k} := \widehat{PATH}_{M,\alpha,k} \wedge \bigwedge_{0 \le i < j \le k} \neg EQ_\alpha(\mathcal{V}_i, \mathcal{V}_j) \qquad (12)$$

Theorem 5. *$\widehat{SIMPLEPATH}_{M,\alpha,k}$ is sat iff $SIMPLEPATH_{\hat{M}_\alpha,k}$ is sat.*

4.3 Abstract Bounded Model Checking

We define the abstract version of the BMC encoding. The formula $\widehat{BMC}_{M,\alpha,k,\varphi}$ is sat iff $BMC_{\hat{M}_\alpha,k,\varphi}$ is sat. Therefore, if $\widehat{BMC}_{M,\alpha,k,\varphi}$ is unsat then we can deduce that there are no initial paths reaching $\hat{\varphi}_\alpha$ in k steps in $\hat{M}_\alpha$. If $\widehat{BMC}_{M,\alpha,k,\varphi}$ is sat, we can extract from its model a satisfying assignment for $BMC_{\hat{M}_\alpha,k,\varphi}$.

Definition 11. *Given an FTS $M = \langle \mathcal{V}, I, T \rangle$, a formula φ, an abstraction function α, and a bound k, the formula $\widehat{BMC}_{M,\alpha,k,\varphi}$ is defined as follows:*

$$\widehat{BMC}_{M,\alpha,k,\varphi} := I(\overline{\mathcal{V}}_0) \wedge EQ_\alpha(\overline{\mathcal{V}}_0, \mathcal{V}_0) \wedge \widehat{PATH}_{M,\alpha,k} \wedge EQ_\alpha(\mathcal{V}_k, \overline{\mathcal{V}}_k) \wedge \varphi(\overline{\mathcal{V}}_k) \qquad (13)$$

Theorem 6. *$\widehat{BMC}_{M,\alpha,k,\varphi}$ is sat iff $BMC_{\hat{M}_\alpha,k,\varphi}$ is sat.*

Similarly, we can define the abstract version of $BMC^{loop}_{M,k}$.

Definition 12. *Given an FTS M, a bound k, and an abstraction function α, the formula $\widehat{BMC}^{loop}_{M,\alpha,k}$ is defined as follows:*

$$\widehat{BMC}^{loop}_{M,\alpha,k} := I(\overline{\mathcal{V}}_0) \wedge EQ_\alpha(\overline{\mathcal{V}}_0, \mathcal{V}_0) \wedge \widehat{PATH}_{M,\alpha,k} \wedge \bigvee_{0\leq l<k,} \bigwedge_{v\in V} EQ_\alpha(\mathcal{V}_l, \mathcal{V}_k) \wedge \bigwedge_{F\in\mathcal{F},} \bigvee_{l\leq h<k} F(V_h) \quad (14)$$

Theorem 7. *$\widehat{BMC}^{loop}_{M,\alpha,k}$ is sat iff $BMC^{loop}_{\hat{M}_\alpha,k}$ is sat.*

4.4 Abstract k-Induction

We define the abstract version of the k-induction conditions. The formulas $\widehat{KINDFW}_{M,\alpha,k}$ and $\widehat{KINDBW}_{M,\alpha,k,\varphi}$ are sat iff respectively $KINDFW_{\hat{M}_\alpha,k}$ and $KINDBW_{\hat{M}_\alpha,k,\varphi}$ are sat. Therefore, if $\widehat{BMC}_{M,\alpha,k,\varphi}$ is unsat and, either $\widehat{KINDFW}_{M,\alpha,k}$ or $\widehat{KINDBW}_{M,\alpha,k,\varphi}$ is unsat then we can conclude that $\hat{\varphi}_\alpha$ is not reachable.

Notice that we do not use the stronger version of $KINDFW_{\hat{M}_\alpha,k}$ and $KINDBW_{\hat{M}_\alpha,k,\varphi}$ defined in [27], because they require to express the negation of $\hat{I}_\alpha$ and $\hat{\varphi}_\alpha$. In fact, the definitions of $\hat{I}_\alpha$ and $\hat{\varphi}_\alpha$ involve an existential quantification, and their negation cannot be handled by the satisfiability solver.

Definition 13. *Given an FTS $M = \langle\mathcal{V}, I, T\rangle$, an abstraction function α, and a bound k, the formula $\widehat{KINDFW}_{M,\alpha,k}$ is defined as follows:*

$$\widehat{KINDFW}_{M,\alpha,k} := I(\overline{\mathcal{V}}_0) \wedge EQ_\alpha(\overline{\mathcal{V}}_0, \mathcal{V}_0) \wedge \widehat{SIMPLEPATH}_{M,\alpha,k} \quad (15)$$

Theorem 8. *$\widehat{KINDFW}_{M,\alpha,k}$ is sat iff $KINDFW_{\hat{M}_\alpha,k}$ is sat.*

Definition 14. *Given an FTS $M = \langle\mathcal{V}, I, T\rangle$, a formula φ, an abstraction function α, and a bound k, the formula $\widehat{KINDBW}_{M,\alpha,k,\varphi}$ is defined as follows:*

$$\widehat{KINDBW}_{M,\alpha,k,\varphi} := \widehat{SIMPLEPATH}_{M,\alpha,k} \wedge EQ_\alpha(\mathcal{V}_k, \overline{\mathcal{V}}_k) \wedge \varphi(\overline{\mathcal{V}}_k) \quad (16)$$

Theorem 9. *$\widehat{KINDBW}_{M,\alpha,k,\varphi}$ is sat iff $KINDBW_{\hat{M}_\alpha,k,\varphi}$ is sat.*

Corollary 2. *If, for all $i \leq k$, $\widehat{BMC}_{M,\alpha,i,\varphi}$ is unsat and, either $\widehat{KINDFW}_{M,\alpha,k+1}$ or $\widehat{KINDBW}_{M,\alpha,k+1,\varphi}$ is unsat, then $\hat{\varphi}_\alpha$ is not reachable in $\hat{M}_\alpha$.*

4.5 Abstract Model Checking

The formulas $\widehat{BMC}_{M,\alpha,k,\varphi}$, $\widehat{BMC}^{loop}_{M,\alpha,k}$, $\widehat{KINDFW}_{M,\alpha,k}$, and $\widehat{KINDBW}_{M,\alpha,k,\varphi}$ can be used to define different procedures for solving reachability and language

```
   input  : concrete FTS $M = \langle \mathcal{V}, I, T \rangle$ and formula $\varphi$
   output : $YES$ if $\hat{\varphi}_\alpha$ is reachable in $\hat{M}_\alpha$, $NO$ otherwise
 1 begin
 2     $k$:=0;
 3     if $\widehat{BMC}_{M,\alpha,k,\varphi}$ is sat then
 4         return $YES$
 5     else if $\widehat{KINDFW}_{M,\alpha,k+1}$ or $\widehat{KINDBW}_{M,\alpha,k+1,\varphi}$ is unsat then
 6         return $NO$
 7     else
 8         k++;
 9 end
```

Algorithm 2. Abstract model checking without abstraction

emptiness of an abstraction of an FTS. We denote such procedures with AMCwa$_{reach}$ and AMCwa$_{loop}$ respectively. AMCwa$_{reach}$ is shown in Algorithm 2. On the lines of k-induction, it iteratively increases the bound k till either it finds a counterexample or it proves the property correct. Unlike Algorithm 1, the path found by BMC is abstract and the bound used by k-induction to conclude is related to the abstract state space. In particular, when the abstract state space is finite, such bound is guaranteed to exist.

AMCwa$_{loop}$ is similar to AMCwa$_{reach}$, but $\widehat{BMC}^{loop}_{M,\alpha,k}$ is used instead of $\widehat{BMC}_{M,\alpha,k,\varphi}$, and only $\widehat{KINDFW}_{M,\alpha,k}$ is used to prove the absence of initial fair paths. In principle, we can add further induction conditions based on fairness, but in practice we experienced that they do not manage to conclude and solve the problem, and therefore we can save the related overhead.

When predicate abstraction is adopted, both AMC and AMCwa are exponential in the number of predicates. However, AMC must find all solutions of the abstraction formula, while AMCwa delegates the blow up to the search. The computation of the abstraction in AMC is orthogonal to the search and is computed upfront, while AMCwa considers only the parts of the abstract state space that are relevant to the search. The solver is used to solve a satisfiability problem rather than to enumerate all possible solutions.

Example 3. Consider the FTS of Ex. 1 and the predicates of Ex. 2. AMCwa proves that the FTS has an empty language by checking that $\widehat{BMC}^{loop}_{M,\alpha,k}$ is unsat for $k = 1, 2, 3, 4$ and that $KINDFW_{M,k}$ is unsat for $k = 4$. Note that k-induction on the concrete FTS cannot prove the same.

5 Experimental Evaluation

5.1 Implementation

We implemented AMCwa$_{reach}$ and AMCwa$_{loop}$ in CEGAR [4], and we evaluated the performance of the two algorithms on an Intel 2.2GHz Laptop equipped with 2GB of memory running Linux. We set a timeout of one hour and the space limit of 2GB. BMC and k-induction problems have been solved with the MathSAT SMT solver without incrementality. The same solver has been used to solve the ALLSAT problem for the computation of the abstraction, as already implemented in CEGAR. All the data and binaries necessary to reproduce the presented results are available at `http://es.fbk.eu/people/tonetta/tests/fm09/`

5.2 Emptiness Checking for Requirements Validation

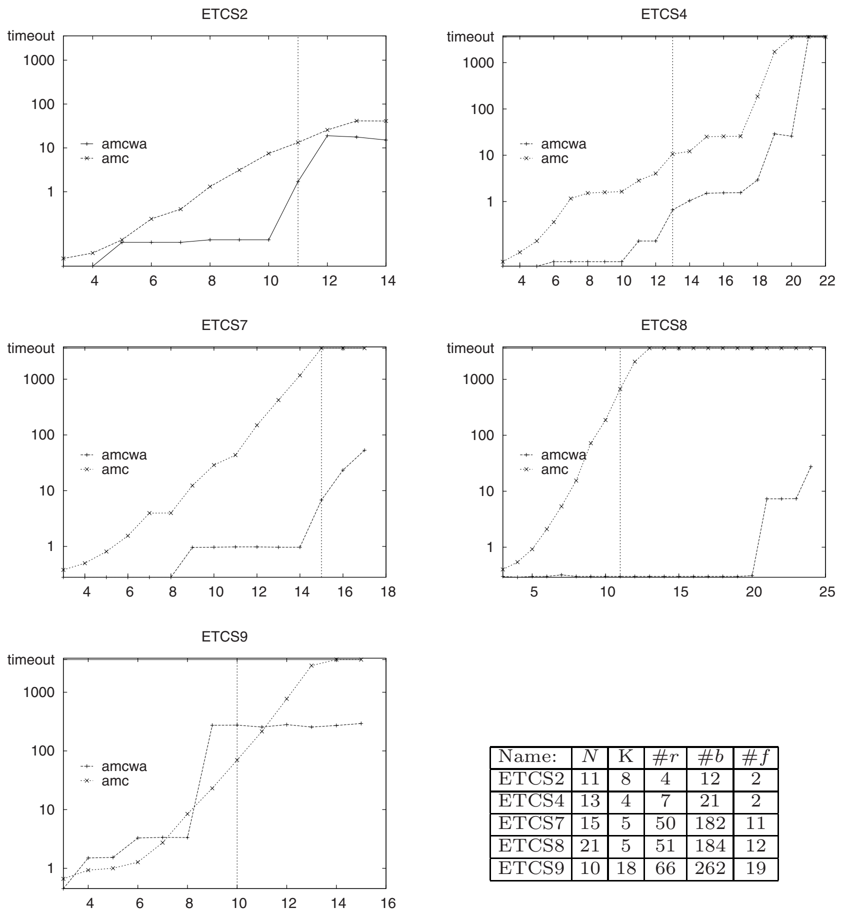
In a project funded by ERA, we investigated the feasibility of the formalization and validation of ETCS functions (see `http://www.era.europa.eu/`). The requirements have been formalized in a fragment of first-order temporal logic [21]. The techniques used to validate the requirements were based on a series of checks for the language emptiness of large transition systems, which encode the consistency of different sets of requirements. The systems had around 300 Boolean variables, 50 real variables, and few integers. Most of times, the requirements were consistent and we used BMC to generate paths as witnesses. Inconsistencies found during the project were mainly due to unfeasible scenarios considered on purpose to test the formalization of the requirements.

We consider the fragment of the ETCS specification analyzed in [6], and we add unfeasible scenarios on the lines of those proposed in the ETCS project. For each problem, we consider a set of predicates that is sufficient to prove the inconsistency. We ran the abstract model checking algorithms with and without the computation of the abstraction for increasing number of predicates.

We obtained the results reported in Fig. 3. The time is plotted in log-scale against the number of predicates. The vertical line highlights the number N which is sufficient to prove the language emptiness of the FTS. Thus, for $i < N$ the algorithm find an abstract (spurious) path, while for $i \geq N$ the algorithm conclude that the language is empty. The tables reports the k at which k-induction stopped with N predicates. The other columns of the table report the size, in terms of number of variables and number of fairness conditions of the FTS. We use $\#r$,$\#b$,$\#f$ with the meaning, r real variables, b Boolean variables, and f fairness conditions.

In ETCS2 and ETCS4, the new algorithm outperforms the computation of the abstraction. In ETCS7 and ETCS8, the improvement scales up exponentially. Note that in ETCS7 and ETCS8, for $i \geq N$, the AMC reaches the timeout. Thus the computation of the abstraction prevents to prove the inconsistency, and the new algorithm manages to prove problems that were not previously solved. Finally, in ETCS9 we have some points were the new algorithm performs worse. However, as the number of predicates scales up the new algorithm is definitely the winner. The data regarding memory consumption have similar plots. As for AMC, the time is almost totally spent in the computation of the abstraction, while the search in the abstract state space is negligible.

Note that, unlike the computation of the predicate abstraction which seems a regular exponential function over the number of predicates, the performance



Name:	N	K	$\#r$	$\#b$	$\#f$
ETCS2	11	8	4	12	2
ETCS4	13	4	7	21	2
ETCS7	15	5	50	182	11
ETCS8	21	5	51	184	12
ETCS9	10	18	66	262	19

Fig. 3. The results of the experimental evaluation

of the new algorithm seems a step function. This is due to the dependency on the depth necessary to use k-induction, and on the fact that some predicates increase such depth, while others do not affect it.

5.3 Number of Predicates vs. Search Depth

We now consider a scalable system manually crafted to investigate the dependency on the number N of predicates and the bound K that is necessary to solve the problem. The system has N Boolean variables P_i, with $1 \leq i \leq N$, and N bounded integers v_i, $1 \leq i \leq N$. The variables P_i are initially *false* and non-deterministically become *true*. For $1 \leq i < N$, the variable P_i can become *true* only if P_{i+1} is *true*. Besides, the variable P_{N-K} can become *true* only if P_1 is *true*. Therefore the property $\neg P_1$ is an invariant of the concrete system, and,

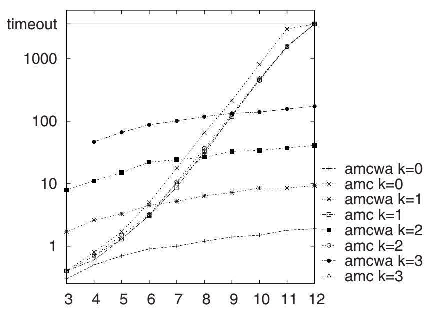


Fig. 4. Results for case study. X axis: parameter N. Y axis: time in seconds.

if we choose $\mathbb{P} = \{P_i\}_{1 \leq i \leq N}$, it is an invariant also of the abstract system. The relationships among the variables P_i involve the variables v_i so that the abstraction does not result straightforward. Moreover, fixed a value for the variables P_i, the variable v_i can range over the whole domain. This makes k-induction of the concrete system infeasible because it would require a too large bound. Note instead that the initial simple paths have at most K steps (basically, the paths that change the variables P_i for $N - K + 1 \leq i \leq N$). This allows us to prove the property on the abstract system by means of k-induction with bound K.

The results are plotted in Fig. 4 in log-scale for increasing values of N. As expected, the time spent in the computation of the abstraction grows exponentially with the number of involved predicates. The search done by AMC$_{reach}$ results to be polynomial in the number of predicates, and strongly depends on the necessary inductive depth.

6 Related Work

Combinations of predicate abstraction with SAT-based techniques are numerous in the literature. As discussed in the introduction, a SAT or SMT solver is typically used to compute the abstraction. The problem of verifying if an abstract path can be simulated on the concrete system is encoded into a BMC problem [9]. Many works on abstraction refinement use also BMC as a model checking procedure. The CEGAR loop described in [20] uses SAT as the only decision procedure. The model checking of the abstract state space is based on BMC and k-induction. Unlike this paper, the abstraction computation and the abstract model checking are distinct steps of the loop.

In [24], BMC is used on the concrete system, the proof of unsatisfiability of the abstract path simulation is used to build the abstraction, and the result of the abstract model checking is used to increase the bound of the search. The work in [16] improves [24] by applying BMC to both the concrete and the abstract system. Also in [22,23], predicates are not used and the abstract system is built by extracting interpolants from the unsatisfiability proof of some path conditions. The efficiency of these works is based on the capability of the refinement to find constraints that are on one hand strong enough to prove the property, on the other hand weak enough to keep the verification complexity low. As the mentioned approaches, this paper aims at avoiding the computation of the abstract state space, but remains in the framework of abstraction precisely defined by a function (rather than computed by the refinement procedure).

In [17], the abstraction is computed on demand along the search. The algorithm exploits the control-flow graph of programs to localize the search to control locations, and avoids building the abstraction for unreachable location. Nevertheless, also this approach needs a quantifier elimination to compute the abstract image of reachable locations.

The work presented in [18] combines symbolic execution with abstraction, but differently from this paper, the abstraction is based on induction and is computed separately from the search. Notably, the counterexamples (which are called *leaping* because they leap due to the abstraction) are used for diagnosis.

A common way to tackle the complexity of predicate abstraction is to approximate the computation by allowing more transitions (see, e.g., [7,12,1]). The complexity is shifted to the refinement that must take care of removing spurious transitions, resulting in an increased number of refinement iterations. This paper focuses only on minimal abstraction, although the technique can be modified in order to search approximated abstract state space.

The definition of $EQ_{\mathbb{P}}$ can be found already in [29], but is used only to compare predicate abstraction with localization reduction.

7 Conclusions

In this paper, we proposed a new algorithm to model check an abstract system without computing the abstraction. While the classic paradigm performs a quantifier elimination to build the abstraction, we encode the model checking problem into satisfiability problems over the concrete variables. We adapted the algorithm based on k-induction to look for finite and infinite fair paths in the abstract system. We showed that the new algorithm can obtain an exponentially better scalability and solved real world problems that were beyond the reach of standard predicate abstraction.

The improvement is of course affected by many parameters. In particular, the abstract state must be amenable for proving the invariant with k-induction. K-induction may be a very effective technique, but, since it is based on the induction principle, it does not manage to prove always an invariant: in the finite-state case, the technique is complete, but the bound necessary to prove the property may become too large; in the infinite-state case, the loop is not guaranteed to terminate. In practice, one has to exploit invariants as in [25].

We plan to integrate the abstract model checking technique into a full abstraction refinement loop. We can exploit the search done by the solver to extract useful information such as abstract transitions, and reachability results on the concrete state space. We can use the obtained leaping counterexamples for diagnosis as suggested in [18]. We can exploit the incrementality of the solver, to boost the search by exploiting the clauses learned by previous iterations and to check the satisfiability of the inductive condition in an incremental way as described in [13].

References

1. Ball, T., Podelski, A., Rajamani, S.K.: Boolean and Cartesian abstraction for model checking C programs. STTT 5(1), 49–58 (2003)
2. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic Model Checking without BDDs. In: Cleaveland, W.R. (ed.) TACAS 1999. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999)
3. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic Model Checking: 10^{20} States and Beyond. Inf. and Comp. 98(2), 142–170 (1992)
4. Cavada, R., Cimatti, A., Franzén, A., Kalyanasundaram, K., Roveri, M., Shyamasundar, R.K.: Computing predicate abstractions by integrating BDDs and SMT solvers. In: FMCAD, pp. 69–76. IEEE, Los Alamitos (2007)
5. Cimatti, A., Roveri, M., Susi, A., Tonetta, S.: Object models with temporal constraints. Journal of Software and Systems Modeling (SoSyM), `http://www.springerlink.com/content/46244553v2769511/`, doi: 10.1007/s10270-009-0130-7
6. Cimatti, A., Roveri, M., Tonetta, S.: Requirements Validation for Hybrid Systems. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 188–203. Springer, Heidelberg (2009)
7. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-Guided Abstraction Refinement. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 154–169. Springer, Heidelberg (2000)
8. Clarke, E.M., Grumberg, O., Long, D.E.: Model Checking and Abstraction. ACM Trans. Program. Lang. Syst. 16(5), 1512–1542 (1994)
9. Clarke, E.M., Gupta, A., Kukula, J.H., Strichman, O.: SAT Based Abstraction-Refinement Using ILP and Machine Learning Techniques. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 265–279. Springer, Heidelberg (2002)
10. Clarke, E.M., Kroening, D., Sharygina, N., Yorav, K.: Predicate Abstraction of ANSI-C Programs Using SAT. FMSD 25(2-3), 105–127 (2004)
11. Colón, M., Uribe, T.E.: Generating Finite-State Abstractions of Reactive Systems Using Decision Procedures. In: Y. Vardi, M. (ed.) CAV 1998. LNCS, vol. 1427, pp. 293–304. Springer, Heidelberg (1998)
12. Das, S., Dill, D.L.: Successive Approximation of Abstract Transition Relations. In: LICS, pp. 51–60 (2001)
13. Eén, N., Sörensson, N.: Temporal induction by incremental SAT solving. Electr. Notes Theor. Comput. Sci. 89(4) (2003)
14. Emerson, E.A., Clarke, E.M.: Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons. Sci. Comput. Program 2(3), 241–266 (1982)
15. Graf, S., Saïdi, H.: Construction of Abstract State Graphs with PVS. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 72–83. Springer, Heidelberg (1997)
16. Gupta, A., Strichman, O.: Abstraction Refinement for Bounded Model Checking. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 112–124. Springer, Heidelberg (2005)
17. Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. In: POPL, pp. 58–70 (2002)

18. Kroening, D., Sharygina, N., Tonetta, S., Tsitovich, A., Wintersteiger, C.M.: Loop Summarization Using Abstract Transformers. In: Cha, S(S.), Choi, J.-Y., Kim, M., Lee, I., Viswanathan, M. (eds.) ATVA 2008. LNCS, vol. 5311, pp. 111–125. Springer, Heidelberg (2008)
19. Lahiri, S.K., Bryant, R.E., Cook, B.: A Symbolic Approach to Predicate Abstraction. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 141–153. Springer, Heidelberg (2003)
20. Li, B., Wang, C., Somenzi, F.: Abstraction refinement in symbolic model checking using satisfiability as the only decision procedure. STTT 7(2), 143–155 (2005)
21. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer, Heidelberg (1992)
22. McMillan, K.L.: Interpolation and SAT-Based Model Checking. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 1–13. Springer, Heidelberg (2003)
23. McMillan, K.L.: Lazy Abstraction with Interpolants. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 123–136. Springer, Heidelberg (2006)
24. McMillan, K.L., Amla, N.: Automatic Abstraction without Counterexamples. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 2–17. Springer, Heidelberg (2003)
25. Pike, L.: Real-time system verification by k-induction. Technical Report TM-2005-213751, NASA Langley Research Center (May 2005)
26. Queille, J.-P., Sifakis, J.: Specification and verification of concurrent systems in CESAR. In: Dezani-Ciancaglini, M., Montanari, U. (eds.) Programming 1982. LNCS, vol. 137, pp. 337–351. Springer, Heidelberg (1982)
27. Sheeran, M., Singh, S., Stålmarck, G.: Checking Safety Properties Using Induction and a SAT-Solver. In: Johnson, S.D., Hunt Jr., W.A. (eds.) FMCAD 2000. LNCS, vol. 1954, pp. 108–125. Springer, Heidelberg (2000)
28. Vardi, M.Y., Wolper, P.: An Automata-Theoretic Approach to Automatic Program Verification. In: LICS, pp. 332–344 (1986)
29. Wang, C., Kim, H., Gupta, A.: Hybrid CEGAR: combining variable hiding and predicate abstraction. In: ICCAD, pp. 310–317 (2007)