Formal Methods in System Design
https://doi.org/10.1007/s10703-023-00434-x

Verification modulo theories

Alessandro Cimatti[1] · **Alberto Griggio**[1] · **Sergio Mover**[2] · **Marco Roveri**[3] · **Stefano Tonetta**[1]

Received: 10 December 2021 / Accepted: 5 May 2023

Abstract
In this paper, we consider the problem of model checking fair transition systems expressed symbolically in the framework of Satisfiability Modulo Theories. This problem, referred to as Verification Modulo Theories, is tackled by combining two key elements from the legacy of Ed Clarke: SAT-based verification and abstraction refinement. We show how fundamental SAT-based algorithms have been lifted to deal with the extended expressiveness with a tight integration of abstraction within a CEGAR loop. In turn, the case of nonlinear theories is based on a CEGAR loop over the linear case. These two elements have also deeply impacted the development of the NuSMV model checker, born from a joint project between FBK and CMU, and its successor nuXmv, whose core integrates SMT-based techniques for VMT.

Keywords Infinite-state transition systems · Formal verification · Model checking · Satisfiability modulo theories · Implicit predicate abstraction

✉ Alessandro Cimatti
cimatti@fbk.eu

Alberto Griggio
griggio@fbk.eu

Sergio Mover
sergio.mover@lix.polytechnique.fr

Marco Roveri
marco.roveri@unitn.it

Stefano Tonetta
tonettas@fbk.eu

[1] Fondazione Bruno Kessler, Trento, Italy

[2] LIX, CNRS, École Polytechnique, Institut Polytechnique de Paris, Palaiseau, France

[3] Department of Information Engineering and Computer Science, University of Trento, Via Sommarive 9, Povo, 38123 Trento, Italy

Published online: 19 September 2023

1 Introduction

In the early 80's, Edmund Clarke coauthored one of the papers that gave birth to the field of Model Checking [1, 2], for which he received the 2007 ACM Turing Award [3]. Substantial breakthrough came from Clarke's work on Bounded Model Checking [4], with two key insights. The first was to consider the problem of debugging, giving up completeness and focusing on the analysis of bounded traces. The second was to understand the potential for SAT-based techniques as a possible replacement of BDDs in symbolic model checking. This paved the way to the development of SAT-based verification techniques of increasing power, from K-induction [5], to interpolation [6] and IC3 [7]. Later, Clarke and his colleagues introduced the fundamental concept of Counter-Example Guided Abstraction Refinement (CEGAR) [8]. CEGAR is an automated approach that combines the idea of reasoning in an abstract space, thus eliminating hopefully irrelevant details, with the idea of selectively introducing additional information based on spurious counterexamples.

These works heavily influenced the field of formal verification, and our own research in the last two decades. In this paper, we give an overview of our work on Verification Modulo Theories (VMT). VMT is the problem of model checking fair transition systems expressed symbolically in the framework of Satisfiability Modulo Theories (SMT). The framework is very appealing, because it is generic: similarly to the case of SMT, the expressiveness of the transition system depends on the background theory. This gives the ability to easily represent various kinds of infinite-state transition systems, and to express temporal properties with a background theory, in particular Linear-time Temporal Logic modulo Theory (LTL(T)). In a nutshell, this research can be seen as generalizing SAT-based verification to the case of infinite-state transition systems, leveraging the enormous progress of the field of SMT, by tightly integrating the computation of effective abstractions.

We first consider the problem of *invariant checking*, by generalizing to the infinite-state case two SAT-based algorithms, K-induction and IC3, by means of predicate abstraction. In particular, we rely on the idea of implicit predicate abstraction [9] to avoid the bottleneck resulting from the eager computation of the abstract space. This enables a tighter integration of the abstraction within a CEGAR loop, as the refinement can be achieved incrementally. The resulting algorithms are able to deal with huge numbers of predicates, which are completely out of reach in the eager case [10].

Then, we consider invariant checking in the presence of *nonlinear theories*. This case can not be easily dealt with by a simple adaptation of the techniques above, since the SMT solvers are, in the case of nonlinear theories, currently unable to fulfill the requirements (interpolation, quantifier elimination) imposed by the algorithms. Hence, we adopt a CEGAR loop where nonlinear operators (e.g. multiplication, transcendental functions) are modeled as uninterpreted functions, and whose interpretation is progressively restricted by means of piecewise-linear constraints [11].

The algorithms for invariant verification lay the foundations to deal with the case of *temporal properties*. Based on the idea of invariant checking, temporal logic model checking is tackled by considering specialized algorithms for the two complementary cases of proving and disproving LTL(T) properties. The first one is based on generalizations of the liveness-to-safety and K-liveness algorithms, with extensions specific to deal with infinite-state transition systems. In [12], implicit predicate abstraction is used to integrate liveness-to-safety and well-founded relations to prove LTL(T) properties. Spurious abstract lasso-shaped counterexamples that are not covered by the well-founded relations are used to find new predicates or new relations in a generalized CEGAR. In [13], K-liveness is

extended for timed systems to avoid spurious counterexamples based on Zeno behaviors. When counting the number of occurrences of a live signal to prove an LTL(T) property, the minimal time between two occurrences is bounded by a symbolic expression derived from the model. The second dual case of finding violations to an LTL(T) property is hindered by the fact that, differently from the finite-state case, false temporal properties may not admit lasso-shaped counterexamples. The idea is hence to provide an underapproximation of the model that only contains counterexample traces. The underapproximation is obtained by a search algorithm that segments the execution trace into a sequence of regions, whose states are eventually "funneled" into the following one, so that progress towards a fair region condition is ensured [14, 15].

All the above algorithms have been implemented in the nuXmv model checker [16], a successor of the NuSMV model checker [17], born from a joint project between FBK and CMU. nuXmv integrates at its core the MathSAT [18] SMT solver to support a number of SMT-based techniques for VMT, and supports the standardized VMT language [19]. In turn, nuXmv is at the core of the HyCOMP [20] model checker for hybrid automata, the xSAP [21] tool for safety assessment, and of the OCRA [22] tool for contract-based design. The techniques described in this paper have been applied in several industrial settings: nuclear [23], biological [24], railways [25, 26], avionics [27], space [28, 29], software engineering [30, 31], and multi-core design [32].

This paper is structured as follows. In Sect. 2, we overview Satisfiability Modulo Theories. In Sect. 3, we define the problem of Verification Modulo Theories. In Sect. 4, we discuss implicit predicate abstraction and refinement. In Sect. 5, we overview the abstraction-based algorithms for invariant checking, and, in Sect. 6, we discuss the extension to the nonlinear case by way of incremental linearization. In Sect. 7, we present the algorithms for LTL(T) model checking. In Sect. 8, we overview the NuSMV and nuXmv verification engines. In Sect. 9 we draw some conclusions.

2 From SAT to SMT

2.1 First-order notation

We work in the setting of standard first order logic. We assume to be given a signature Σ of function and predicate symbols. A 0-ary function symbol is called a *constant*. A Σ-*term* is a first-order term built out of function symbols and variables. If $t_1, \ldots, t_n$ are Σ-terms and p is a predicate symbol, then $p(t_1, \ldots, t_n)$ is a Σ-*atom*. A Σ-*formula* ϕ is built in the usual way out of the universal and existential quantifiers, Boolean connectives, and Σ-atoms. When Σ is implicit, we omit it and just talk about terms, atoms, and formulas. A *literal* is either an atom or its negation. We call a formula *quantifier-free* if it does not contain quantifiers, and *ground* if it does not contain free variables.

A *clause* is a disjunction of literals. A formula is said to be in *conjunctive normal form* (CNF) if it is a conjunction of clauses. For every non-CNF formula φ, an equisatisfiable CNF formula ψ can be generated in polynomial time [33].

We also assume the usual first-order notions of interpretation, satisfiability, validity, logical consequence, and theory, as given, e.g., in [34]. We write $\Gamma \models \phi$ to denote that the formula ϕ is a logical consequence of the (possibly infinite) set Γ of formulas. A *first-order theory*, $\mathcal{T}$, is a set of first-order sentences. A structure $\mathcal{A}$ is a model of a theory $\mathcal{T}$ if $\mathcal{A}$

satisfies every sentence in $\mathcal{T}$. A formula is *satisfiable in* $\mathcal{T}$ (or $\mathcal{T}$ *-satisfiable*) if it is satisfiable in a model of $\mathcal{T}$.

In what follows, with a little abuse of notation, we might denote conjunctions of literals $l_1 \wedge \ldots \wedge l_n$ as sets $\{l_1, \ldots, l_n\}$ and vice versa. If $\eta \equiv \{l_1, \ldots, l_n\}$, we might write $\neg\eta$ to mean $\neg l_1 \vee \ldots \vee \neg l_n$. Moreover, following the terminology of the SAT and SMT communities, we shall refer to predicates of arity zero as *propositional variables*, and to uninterpreted constants as *theory variables*. Finally, if a formula φ is satisfiable, we shall call a *model* of φ any assignment μ to (possibly a subset of) the variables of φ and interpretation of symbols which make the formula true, and we denote this with $\mu \models \varphi$. If μ is a model and x is a variable, we write $\mu[x]$ for the value of x in μ.

Note that we use the symbol $\models$ with different denotations. If ϕ and ψ are state formulas, $\phi \models \psi$ denotes that ψ is a logical consequence of ϕ. If μ is an interpretation, $\mu \models \psi$ denotes that μ is a model of ψ. Later, we will introduce transition systems, paths, invariant properties and LTL properties. If S is a transition system and ψ an invariant property, $S \models \psi$ denotes that ψ is an invariant of S. If instead ψ is a LTL($\mathcal{T}$) formula and σ an infinite-path of the transition system S, we use $\sigma \models \psi$ to denote that the path σ satisfies the LTL($\mathcal{T}$) formula ψ and $S \models \psi$ to denote that all the infinite paths of S satisfy ψ. Different usages of $\models$ will be clear from the context.

2.2 SAT and SMT solvers

A *SAT solver* is a procedure that can decide the satisfiability of a propositional formula φ (typically assumed to be in CNF), i.e. it returns true iff φ has at least one model. Modern SAT solver implementations typically follow the CDCL (Conflict-Driven-Clause-Learning) architecture [35]. Given a first-order theory $\mathcal{T}$, an *SMT solver* for $\mathcal{T}$ – *SMT*($\mathcal{T}$)– is a procedure that is able to decide the satisfiability of *Boolean combinations* of (quantifier-free) propositional atoms and theory atoms in $\mathcal{T}$.[1] Examples of useful theories are equality and uninterpreted functions, difference logic and linear arithmetic, either over the rationals or the integers, the theory of arrays, that of bit vectors, and their combinations.

The currently most popular approach for solving the *SMT*($\mathcal{T}$) problem is the so-called *"lazy"* approach [36, 37], also frequently called DPLL($\mathcal{T}$) or CDCL($\mathcal{T}$) [38]. The lazy approach works by combining a propositional SAT solver based on the CDCL algorithm with a *theory solver for* $\mathcal{T}$ ($\mathcal{T}$-solver), which is a procedure that can decide the satisfiability in $\mathcal{T}$ of sets/conjunctions of ground atomic formulas and their negations. Essentially, CDCL is used as an enumerator of truth assignments μ_i of the atoms of φ that propositionally satisfy the input formula, and the $\mathcal{T}$-solver is used for checking the $\mathcal{T}$-satisfiability of the enumerated assignments: if the current μ_i is $\mathcal{T}$-satisfiable, then φ is $\mathcal{T}$-satisfiable. Otherwise, if μ_i is $\mathcal{T}$-unsatisfiable, the $\mathcal{T}$-solver generates an *unsatisfiable core* η_i, i.e. a subset of μ_i which is still $\mathcal{T}$-unsatisfiable; the negation of η_i, which is a clause that is *valid* in $\mathcal{T}$ (and therefore called a *theory lemma or* $\mathcal{T}$*-lemma*), is then added to the input formula, and the CDCL solver is called again on the updated formula. The procedure continues until either a $\mathcal{T}$-satisfiable μ_i is found, or the problem becomes propositionally unsatisfiable (and therefore also $\mathcal{T}$-unsatisfiable). In practice, several optimizations and heuristics are applied

[1] Here we are implicitly assuming that the ground satisfiability problem for $\mathcal{T}$ is decidable. We shall relax this assumption in Sect. 6.

to make this basic description of the lazy SMT approach competitive. We refer the interested reader to [36, 37] for more details.

3 Verification modulo theories

3.1 Symbolic fair transition systems

We represent infinite-state systems over a background theory $\mathcal{T}$ with signature Σ. A *transition system* S is a tuple $\langle X, I, T\rangle$ where X is a set of (state) variables, $I(X)$ is a Σ-formula representing the initial states, and $T(X, X')$ is a Σ-formula representing the transitions ($X' := \{x' \mid x \in X\}$ is the set of variables representing the next state of the transition system).

A *state* s of a transition system S is an interpretation $\langle \mathcal{M}, \mu_s\rangle$ to the symbols in the signature Σ and the variables X, where $\mathcal{M}$ and μ_s are respectively the domain and the assignment of the interpretation. The satisfaction relation $s \vDash \phi$ for a Σ-formula is defined as usual. We write s' for the state s where the assignments to the variables $x \in X$ from s are substituted with assignments to the variables $x' \in X'$ (i.e., for all $x \in X$, $s(x) = s'(x')$). A *finite path* (of length k) of S is a finite sequence $\pi := s_0, s_1, \ldots, s_k$ of states with the same domain and interpretation of the symbols in Σ (e.g., for a term $t \in \Sigma$, $\mu_{s_i}[t] = \mu_{s_j}[t]$, for any index i, j) such that $s_0 \vDash I$, and for all i, $0 \leq i < k$, $s_i, s'_{i+1} \vDash T$. Notice that the interpretation of the symbols in the signature Σ is *rigid*, meaning that the interpretation to uninterpreted functions and predicates does not change across the states in a path (while the assignments to the variables X can change). We say that a state s is *reachable* in S if and only if there exists a path of S ending in s. The set of reachable states of a transition system represented with formulas interpreted over theories can be infinite. We denote the set of all states (not necessarily reachable) of a system with state variables X as $\mathcal{S}_X$.

Example 1 $S = \langle \{c, d\}, c = 0 \wedge d = 0, c' = c + d \wedge d' = d + 1\rangle$ is an infinite-state transition system, where $\{c, d\}$ are integer variables. The initial state s_0 of the system has an assignment μ_{s_0} where $\mu_{s_0}[c] = 0$ and $\mu_{s_0}[d] = 0$. At every transition, the system S increases d by one and increases c by d. A path $\pi := s_0, s_1, s_2$ of the system S is such that $\mu_{s_0} \vDash c = 0 \wedge d = 0$, $\mu_{s_1} \vDash c = 0 \wedge d = 1$, $\mu_{s_2} \vDash c = 1 \wedge d = 2$.

A *fair transition system* $S := \langle X, I, T, F\rangle$ is a transition system with an additional fairness condition F. An *infinite path* $\sigma := s_0, s_1, \ldots$ of a fair transition system S is such that all the states have the same domain and interpretation of symbols in Σ, $s_0 \vDash I$ and for all $i > 0$ $s_{i-1}, s'_i \vDash T$. An infinite path is a *fair path* if for each $i \geq 0$ there exists $j > i$ such that $s_j \vDash F$ (i.e., the path visits the fairness condition F infinitely often).

Example 2 Consider the fair transition system $S = \langle \{c, d\}, c = 0 \wedge d = 0, (c' = 0 \wedge d' = 0) \vee (c' = c + d \wedge d' = d + 1), c > d\rangle$, which has a similar transition relation to the transition system in example 1. The infinite path that keeps incrementing the value of the c and d variables is a fair path (e.g., the path with the assignments $\{c = 0, d = 0\}, \{c = 0, d = 1\}, \{c = 1, d = 2\}, \{c = 3, d = 3\}, \ldots$), while there are some infinite paths that are not fair (e.g., the path where c and d never change value $\{c = 0, d = 0\}, \{c = 0, d = 0\}, \{c = 0, d = 0\}, \ldots$), since they don't reach a state that satisfies $c < d$ infinitely often.

Given two (fair) transition systems $S_1 := \langle X_1, I_1, T_1, F_1 \rangle$ and $S_2 := \langle X_2, I_2, T_2, F_2 \rangle$, their synchronous product is $S_1 \times S_2 := \langle X_1 \cup X_2, I_1 \wedge I_2, T_1 \wedge T_2, F_1 \wedge F_2 \rangle$.

In the following, we implicitly assume to represent the transition systems over a theory $\mathcal{T}$ with a signature Σ.

3.2 LTL($\mathcal{T}$): linear temporal logic modulo theory

Linear-time Temporal Logic (LTL) [39] is a modal logic to express sets of infinite traces of a transition system. We consider LTL($\mathcal{T}$) formulas [40],[2] an extension of LTL where atomic propositions are first-order predicates over the theory $\mathcal{T}$ and with variables X and X'. Formally, a LTL($\mathcal{T}$) formula ϕ is:

$$\phi := p(t_1, \ldots, t_k) \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid \mathbf{X}\phi_1 \mid \phi_1 \mathbf{U} \phi_2,$$

where p is a predicate, ϕ_1 and ϕ_2 are LTL($\mathcal{T}$) formulas, and $t_1, \ldots, t_k$ are terms. A term t is:

$$t := f(t_1, \ldots, t_k) \mid a \mid x \mid x',$$

where f is a function symbol, $t_1, \ldots, t_k$ are terms, a is a constant, $x \in X$ is a variable (in a set of variables X), and $x' \in X'$ is a variable in the set of variables representing the next state. A predicate of LTL($\mathcal{T}$) also contains the next state variables X' as terms. In such way, a LTL($\mathcal{T}$) formula can predicate about the next value of a variable, instead of just relating formulas in the current state and next state of the path with the LTL temporal operator $\mathbf{X}$. We will use the standard abbreviations $\mathbf{F}\phi := \top \mathbf{U} \phi$ and $\mathbf{G}\phi := \neg\mathbf{F}\neg\phi$ to denote the "finally" $\mathbf{F}$ and "globally" $\mathbf{G}$ operator.

In the following we consider infinite paths where all the states have the same domain and interpretation of symbols in Σ, while the assignments to the set of variables X in the states can change. Given an infinite path $\sigma := s_0, s_1, \ldots$ we write $\sigma[i]$ for the i-th element of σ (i.e., s_i) and we write σ^i for the suffix of σ starting from state i-th (i.e., $\sigma^i := s_i, s_{i+1}, \ldots$). We define when an infinite path σ satisfies the LTL($\mathcal{T}$) formula ϕ (i.e., $\sigma \vDash \phi$) by induction:

- $\sigma \vDash p(t, \ldots, t)$ iff $\sigma[0], \sigma[1]' \vDash p(t, \ldots, t)$;
- $\sigma \vDash \neg p(t, \ldots, t)$ iff $\sigma[0] \nvDash p(t, \ldots, t)$;
- $\sigma \vDash \phi_1 \wedge \phi_2$ iff $\sigma \vDash \phi_1$ and $\sigma \vDash \phi_2$;
- $\sigma \vDash \mathbf{X}\phi$ iff $\sigma^1 \vDash \phi$;
- $\sigma \vDash \phi_1 \mathbf{U} \phi_2$ iff for some $j \geq 0$, $\sigma^j \vDash \phi_2$ and for all $0 \leq k < j$, $\sigma^k \vDash \phi_1$.

Example 3 Consider for example the LTL formula $\mathbf{FG}\, c < d$ and the path $\sigma = \{c = 0, d = 0\}, \{c = 0, d = 1\}^\omega$. The path satisfies the formula because it assigns initially both c and d to 0, and then c to 0 and d to 1 forever.

[2] The work in [40] further introduces event freezing operators "at next" and "at last", which we do not consider here for simplicity.

3.3 Problem definition

3.3.1 Invariant checking

Given a transition system $S = \langle X, I, T \rangle$ and formula $P(X)$ over the variables X, the *invariant verification problem* for S and P is the problem to check if all the reachable states of S satisfy the formula P. In that case, we say that the transition system S satisfies the formula P, written as $S \models P$. The formula $P(X)$ represents a set of "safe" states of the system, while any state in the complement $\neg P$ is an "unsafe" state. The dual formulation of the invariant verification problem is the *reachability* problem, which asks if there exists a reachable state s such that $s \models \neg P$.

A challenge when checking invariant properties for infinite-state systems is to compute a possibly infinite set of reachable states. Instead, a central notion to solve the invariant verification problem for infinite-state systems are inductive invariants: a formula ψ is an *inductive invariant* for a transition system S if $I(X) \models \psi(X)$ and $\psi(X) \wedge T(X, X') \models \psi(X')$. We can solve the invariant verification problem by finding an inductive invariant ψ such that $\psi \models P$ (i.e., if ψ is an inductive invariant and $\psi \models P$, then $S \models P$). Note that the set of reachable states $R(X)$ is the strongest inductive invariant (i.e., $R(X)$ is an inductive invariant and for any other inductive invariant $\psi(X)$ we have that $R(X) \models \psi(X)$).

3.3.2 LTL($\mathcal{T}$) model checking

The *LTL($\mathcal{T}$) model checking problem* is the problem of checking if all the infinite paths σ of a fair transition system $S := \langle X, I, T \rangle$ satisfy an LTL($\mathcal{T}$) formula ϕ (i.e., for all infinite paths σ of S, $\sigma \models \phi$). We denote the LTL($\mathcal{T}$) verification problem as $S \models \phi$.

In the following, we consider standard reductions [41–43] of the LTL model checking problem $S \models \phi$ to the model checking problem $S \times S_{\neg\phi} \models \mathbf{FG}\neg f_{\neg\phi}$ (i.e., $S \models \phi$ if and only if $S \times S_{\neg\phi} \models \mathbf{FG}\neg f_{\neg\phi}$), where $S_{\neg\phi}$ is a transition system with a fairness condition $f_{\neg\phi}$. Such reduction allows us to prove that $S \not\models \phi$ by finding a counterexample in the form of a fair path, i.e., a path that visits the fairness condition $f_{\neg\phi}$ of $S \times S_{\neg\phi}$ infinitely many times. In the following, we assume that the above transformation has been applied to the LTL($\mathcal{T}$) model checking problem, and consider a problem in the form $S \models \mathbf{FG}\neg f_{\neg\phi}$, where ϕ is a formula whose atoms are either propositional variables or predicates. When clear from the context, we also drop the subscript $\cdot_{\neg\phi}$, and simply use $\mathbf{FG}\neg f$.

3.4 Algorithms for model checking modulo theories

Symbolic transition systems represented with propositional logic formulas can model finite-state systems such as hardware design. Several successful symbolic model checking algorithms [4–7] for finite-state systems are based on SAT solvers. Furthermore, as we highlighted in Sect. 2, in the last two decades there has been an important progress in deciding the satisfiability of formulas in first-order logic modulo theory. Such improvements enable a seamless transition of the verification algorithms for finite-state systems based on SAT solvers to algorithms that verify infinite-state systems, represented with SMT formulas, based on SMT solvers. In the following, we survey how the main verification algorithms that have been proposed in the SAT-based setting can

be naïvely applied to infinite-state systems by simply replacing the underlying decision procedure (i.e., from a SAT to an SMT solver).

3.4.1 Invariant checking

Bounded Model Checking (BMC). BMC [4] is a symbolic reachability analysis algorithm that explores paths of a transition system that can be reached in a bounded number of steps k. BMC was first introduced to find violations to LTL properties for finite-state systems and was subsequently extended to infinite-state systems [44], substituting the underlying SAT solver with a SMT solver. BMC encodes the problem of finding a violation to an invariant property P with a path of length k as follows:

$$BMC^k := I(X^0) \wedge \bigwedge_{0 \leq i < k} T(X^i, X^{i+1}) \wedge \neg P(X^k), \tag{1}$$

where $X^i := \{x^i \mid x \in X\}$ is a set of copies of the set of variables X indexed with $i \in \mathbb{N}$. The formula BMC^k is satisfiable if and only if there exists a finite path π of the transition system S that reaches a state s_k in k steps and $s_k \nvDash P$. To find a violation, we encode the condition BMC^k for an increasing value of k, and check each time the satisfiability of the formula BMC^k. Note that different encodings of the BMC problems are possible (e.g., one may encode the problem of finding violations in any of the i-steps from 0 to k).

-K-Induction. While BMC could, in principle, prove that $S \vDash P$ by exploring a sufficiently large bound k, such an upper bound is generally very large for finite-state systems and does not exist, in general, for infinite-state systems. Thus, in practice BMC is effective only in finding violations to an invariant verification problem. K-induction [5] builds on top of BMC and generalizes the induction principle to multiple steps of the system with the goal of proving an invariant property. The k-induction proof consists of a base and an inductive step. The base step proves that an invariant property P holds for all the states reachable in $k-1$ steps. We prove this step showing that a BMC query similar to Eq. (1), but where the violation of the property is checked at every time step (i.e., $\bigvee_{i=0}^{k} \neg P(X^i)$), is unsatisfiable. The inductive step proves either that: any path of length k cannot reach a state that was not visited with a path of length $k-1$ (Eq. (2)); or any safe path of length $k-1$ cannot be extended to a path of length k that violates P (Eq. (3)). The formulas that formalize the inductive steps are:

$$KINDFW^k := I(X^0) \wedge SIMPLE^k, \tag{2}$$

$$KINDBW^k := SIMPLE^k \wedge \neg P(X^k), \tag{3}$$

where:

$$SIMPLE^k := \bigwedge_{0 \leq i < k} T(X^i, X^{i+1}) \wedge \bigwedge_{0 \leq i < k} P(X^i) \wedge \bigwedge_{0 \leq i < j \leq k} X^i \neq X^j, \tag{4}$$

and $X^i \neq X^j := \bigvee_{x \in X} x^i \neq x^j$. The simple path formula $SIMPLE^k$ restrict the search to simple paths, paths containing all different states (i.e., paths that never visit a state more than once). The formula $KINDFW^k$ encodes the set of simple paths of length k starting from an initial state of S. If such formula is unsatisfiable, then there is no path π of length k that contains an unseen state in a path of length $k-1$. Thus, if the base case holds (i.e., all the states visited in the system up to length $k-1$ satisfy P) and the formula $KINDFW^k$

is unsatisfiable, we can conclude that $S \vDash P$ since we visited all the states of S. The formula $KINDBW^k$ provides another sufficient condition to prove that $S \vDash P$ and encodes that an (uninitialized) path of length $k-1$ that is safe can be extended to a path of length k that can violate P. Also in this case, the simple path formula $SIMPLE^k$ encodes the uniqueness of the states in the suffix. If the base case holds and the formula $KINDBW^k$ is unsatisfiable we can conclude that $S \vDash P$.

IC3. IC3[7] is an efficient SAT-based algorithm for the verification of finite-state systems, with Boolean state variables and propositional logic formulas. The IC3 algorithm tries to prove that $S \vDash P$ by finding a suitable inductive invariant $\mathcal{F}(X)$ such that $\mathcal{F}(X) \vDash P(X)$. In order to construct $\mathcal{F}$, IC3 maintains a sequence of formulas (called *trace*) $\mathcal{F}_0(X), \ldots, \mathcal{F}_k(X)$ such that: (i) $\mathcal{F}_0 = I$; (ii) $\mathcal{F}_i \vDash \mathcal{F}_{i+1}$; (iii) $\mathcal{F}_i(X) \wedge T(X, X') \vDash \mathcal{F}_{i+1}(X')$; (iv) for all $i < k$, $\mathcal{F}_i \vDash P$. Therefore, each element of the trace $\mathcal{F}_{i+1}$, called a *frame*, is inductive relative to the previous one, $\mathcal{F}_i$. IC3 strengthens the frames by finding new relative inductive clauses. A clause c is inductive relative to the frame $\mathcal{F}$, i.e. $\mathcal{F} \wedge c \wedge T \vDash c'$, iff the formula

$$RelInd(\mathcal{F}, T, c) \quad := \quad \mathcal{F} \wedge c \wedge T \wedge \neg c', \tag{5}$$

is unsatisfiable, so that a check of relative inductiveness can be directly tackled by a SAT solver.

At a high level, IC3 proceeds incrementally by alternating two phases: a blocking phase, and a propagation phase. In the *blocking* phase, the trace is analyzed to prove that no intersection between $\mathcal{F}_k$ and $\neg P(X)$ is possible. During this phase, the trace is enriched with additional formulas, which can be seen as strengthening the approximation of the reachable state space. At the end of the blocking phase, either $\mathcal{F}_k \vDash P$ is proved or a counterexample is generated.

In the *propagation* phase, IC3 tries to extend the trace with a new formula $\mathcal{F}_{k+1}$, moving forward the clauses from preceding $\mathcal{F}_i$'s. If, during this process, two consecutive frames become identical (i.e. $\mathcal{F}_i = \mathcal{F}_{i+1}$), then a fixpoint is reached, and IC3 terminates with $\mathcal{F}_i$ being an inductive invariant proving the property.

In the blocking phase IC3 maintains a set of pairs (s, i), where s is a set of states that can lead to a bad state, and $i > 0$ is a position in the current trace. New formulas (in the form of clauses) to be added to the current trace are derived by (recursively) proving that a cube s of a pair (s, i) is unreachable starting from the formula $\mathcal{F}_{i-1}$. This is done by checking the satisfiability of the formula $RelInd(\mathcal{F}_{i-1}, T, \neg s)$. If the formula is unsatisfiable, then $\neg s$ is *inductive relative to* $\mathcal{F}_{i-1}$, and the bad state s can be *blocked* at i. This is done by *generalizing* $\neg s$ to a *stronger clause* $\neg g$ that is still inductive relative to $\mathcal{F}_{i-1}$, and adding $\neg g$ to $\mathcal{F}_i$. Inductive generalization is a central step of IC3, that is crucial for the performance of the algorithm. Adding $\neg g$ to $\mathcal{F}_i$ blocks not only the bad cube s, but possibly also many others, thus allowing for a faster convergence of the algorithm.

If, instead, (5) is satisfiable, then the overapproximation $\mathcal{F}_{i-1}$ is not strong enough to show that s is unreachable. In this case, let p be a subset of the states in $\mathcal{F}_{i-1} \wedge \neg s$ such that all the states in p lead to a state in s' in one transition step. Then, IC3 continues by trying to show that p is not reachable in one step from $\mathcal{F}_{i-2}$ (that is, it tries to block the pair $(p, i-1)$). This procedure continues recursively, possibly generating other pairs to block at earlier points in the trace, until either IC3 generates a pair $(q, 0)$, meaning that the system does not satisfy the property, or the trace is eventually strengthened so that the original pair (s, i) can be blocked.

3.4.2 Liveness checking

Encoding lasso-shaped paths. The liveness verification problem $S \vDash \mathbf{FG}\neg f$ amounts to show that no infinite path in S visits f infinitely often. We can modify the BMC encoding to find *lasso-shaped* paths. A path $\pi := s_0, s_i, \dots s_k$ is lasso-shaped if there exists $j < k$ such that $s_k, s_j \vDash T(X^k, X^j)$ (i.e., the state s_k can "loop-back" to the state s_j). The path is formed by a finite path $s_0, \dots, s_{j-1}$ and a suffix $s_j, \dots, s_k$, which represents a loop. A lasso-shaped path represents the infinite path of the system where the suffix can be repeated for an infinite number of times. The BMC encoding (1) can be modified to find fair lasso-shaped paths as follows:

$$BMC_f^k := I(X^0) \wedge \bigwedge_{0 \le i < k} T(X^i, X^{i+1}) \wedge \bigvee_{0 \le j < k} \left(\bigwedge_{x \in X} x^k = x^j \wedge \bigvee_{j \le z < k} f^z \right). \tag{6}$$

The formula BMC_f^k is satisfiable if there exists a lasso-shaped path where at least one of the states in the loop satisfies f.

Liveness to safety reduction. The *liveness to safety reduction* (L2S) [45] encodes the liveness model checking problem as an invariant model checking problem. The L2S encoding transforms the transition system S to the new transition system $S_{L2S} = \langle X \cup X_{L2S}, I_{L2S}, T_{L2S} \rangle$, where the set of variables X_{L2S} contains a copy of the system variables and the additional variables $\{seen, triggered, loop\}$. The new transition relation T_{L2S} guesses non-deterministically a state of the system where a loop of a fair path starts and stores such state in the variables X_{L2S}. When doing so, it also sets the Boolean variable *seen* to true for recording that a loop started. The transition relation further records (with the *triggered* variable) if the fairness condition f has been seen since the start of the loop, and records if a fair lasso-shaped path for the fairness condition f exists with the *loop* variable. In the case S is a finite-state system, the reduction is such that $S \vDash \mathbf{FG}\neg f$ if and only if $S_{L2S} \vDash \neg loop$. In the infinite-state case, instead, the *L2S* reduction can only find some of the violations to the liveness properties for the system S, i.e., if $S_{L2S} \nvDash \neg loop$ then $S \nvDash \mathbf{FG}\neg f$, but the converse does not hold.

K-Liveness. K-Liveness [46] is an algorithm that reduces the liveness verification problem $S \vDash \mathbf{FG}\neg f$ to a sequence of safety verification problems. The main observation of the K-Liveness algorithm is that each path σ of the system S satisfies f a finite number of times iff $S \vDash \mathbf{FG}\neg f$. The K-Liveness algorithm finds a non-negative upper bound K on the number of times a path of S visits a state s that satisfy f. Let the formula $\#(f) \ge K$ be true for a finite path π of S, written as $\pi \vDash \#(f) \ge K$, if the number of times a state satisfying f in the path π is not greater than K (i.e., $\#(f)$ denotes the size of the set $\{i \mid \pi[i] \vDash f\}$). We express that all the paths of the system S do not reach f more than K times with $S \vDash \#(f) \le K$. We have that:

$$\text{if } \exists K \in \mathbb{N}. S \vDash \#(f) \le K \text{ then } S \vDash \mathbf{FG}\neg f. \tag{7}$$

The K-Liveness algorithm iteratively finds such K by solving a sequence of verification problems for an increasing value of K (i.e., the algorithm checks $S \vDash \#(f) \le 0$, $S \vDash \#(f) \le 1$, ...).

We can easily reduce each verification problem $S \vDash \#(f) \leq n$, for a natural number $n \in \mathbb{N}$, to an *invariant verification problem*. We first augment the transition system S with an additional variable c counting the number of times a state satisfies f (i.e., the variable c starts from 0 and increments its value by 1 whenever the system visits a state satisfying f). Then, we verify that $S \vDash c \leq n$ with an invariant model checking algorithm. In the original paper proposing K-Liveness [46] the authors propose an efficient implementation that uses the IC3 algorithm to solve each safety verification problem. Such implementation exploits the incremental nature of IC3 to reuse all the frames IC3 learned when solving the problem $S \vDash c \leq n$ to solve the next verification problem $S \vDash c \leq n + 1$. This optimization is sound since the transition system S does not change and the property $c \leq n$ implies $c \leq n + 1$, and both conditions ensure that the invariants on the frames learned when verifying $S \vDash c \leq n$ also hold when verifying $S \vDash c \leq n + 1$.

3.5 Challenges when verifying infinite-state systems

The algorithms we present above are *sound* when we apply them to a symbolic fair transition system and we use SMT to decide satisfiability. However, since both the invariant and liveness verification problems are undecidable for infinite-state transition systems, the procedures may not terminate. Furthermore, the procedures above may also not terminate on decidable subclasses of the verification problem (e.g., safety verification timed automata [47]) or on specific problem instances that may have a solution (e.g., k-induction and IC3 may be not be able to find an inductive invariant even if one exists). Here, we focus our attention on the second problem, and we redirect the reader to several of the ad-hoc extensions of the above techniques that target decidable cases for a description of solutions to the first one (e.g. [48, 49]).

Invariant checking. In the infinite-state setting, Bounded Model Checking, k-induction, and IC3 will eventually find a violation to a safety property, if such a violation exists. However, in an infinite-state system there is no maximum depth k that guarantees to visit all the reachable states (the least number of steps to reach all the reachable states of in an infinite state system, the *diameter*, can be unbounded).

Both the k-induction and IC3 algorithms may fail to prove that a property holds. In k-induction, the inductive check with k steps may not be sufficient to prove the system is safe, and the simple path condition $SIMPLE^k$ can be satisfiable for any k, preventing both formulas $KINDFW^k$ and $KINDBW^k$ from ever being unsatisfiable. A naïve extension of the IC3 algorithm to SMT is not effective because of the blocking phase. When the algorithm cannot block a pair (s, i), it finds a predecessor state $(p, i-1)$ to block from an assignment μ satisfying the relative induction formula (5) (i.e., $RelInd(\mathcal{F}_{i-1}, T, \neg s)$). Such strategies would block a single state from an infinite set of states, and thus is ineffective. In Sect. 5 we present modifications to the k-induction and IC3 algorithms that efficiently verify a sequence of finite-state abstractions of the system using counterexample-guided abstraction refinement (CEGAR) [50].

Example 4 Both k-induction and IC3 will fail to verify that the transition system $S = \langle \{c, d\}, c = 0 \wedge d = 0, (c' = c + d \wedge d' = d + 1) \rangle$ from the Example 1 satisfies the property $d \leq 3 \vee c > d$. Observe that such property holds, since when $d = 4$ the variable c is 4, and then we have that $c > d$ in the following states of the path.

Applying k-induction would fail: (i) The base step for any $k \geq 0$ succeeds (cleary, BMC cannot find a counterexample); (ii) the $KINDFW^k$ formula (2) is always satisfiable, since every path of length $k-1$ can be extended to a k-long path with an "unseen" state (e.g., incrementing d by one will obtain a state with a new value of d for the path); (iii) similarly, the $KINDBW^k$ formula (3) is also satisfiable following a similary reasoning.

Applying a naïve version of IC3 would also be ineffective: the algorithm would keep enumerating states that do not satisfy $d \leq 3 \vee c > d$ in the blocking phase. While each one of such states can be blocked by the previous frame via the the relative inductive check (5), their number is still infinite, so the algorithm would not terminate.

Liveness checking. One of the main challenges when verifying liveness properties for infinite-state systems is that a violated property is not guaranteed to have a lasso-shaped counterexample (e.g., all counterexamples may be paths where the value of variables diverge).

The BMC encoding shown in (6) only finds lasso-shaped paths, so the algorithm is not guaranteed to find a counterexample, even when one exists. The algorithm in Sect. 7.3 addresses this issue using the notion of recurrence sets.

Example 5 Consider the fair transition system $S = \langle \{c, d\}, c \geq 0, c' = c + d \wedge d' = d + 1, \top \rangle$ and the LTL($\mathcal{T}$) property $\mathbf{G}\, c \leq d$. Clearly, $S \not\models \mathbf{G}\, c \leq d$. However, BMC would fail since the transition system has *only* fair paths that are not lasso-shaped.

The existence of non lasso-shaped paths does not allow to apply the same liveness to safety [45] reduction for finite-state systems for *proving* a liveness property. The reduction works by recoding the occurrence of a lasso-shaped path violating the fairness condition, so such reduction does not take into account non lasso-shaped paths. In Sect. 7.1 we describe an algorithm that circumvents such issues using abstraction techniques and well-founded relations.

Example 6 Consider the fair transition system $S = \langle \{c, d\}, c \geq 0, c' = c + d \wedge d' = d + 1, \top \rangle$ and the LTL($\mathcal{T}$) property $\mathbf{FG}\, c < d$. We have that $S \not\models \mathbf{FG}\, c < d$ since, independently from the initial value of d, d will eventually be positive and c will eventually become greater than d. The transition system has only fair paths that are not lasso-shaped. Such paths are ignored by the L2S reduction that, when applied naïvely, results in a transition system where the *loop* variable is never true (i.e., there are no fair paths in the L2S reduction). In this case, verifying the L2S reduction would wrongly conclude that $S \models \mathbf{FG}\, c < d$.

The existence of paths that are not lasso-shaped further affects k-liveness. K-liveness proves fairness by showing that there exists a bound on the number of times a fairness property is falsified. Even if a fairness property holds, such bound may not be an integer number, but a value that depends on a, possibly infinite-valued, variable of the transition system (e.g., an uninitialized parameter). In Sect. 7.2 we present an algorithm that tackles such issues in transition systems where the value of a variable diverges along all the computation paths.

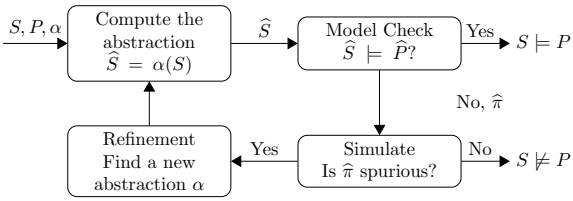


Fig. 1 The CEGAR loop for the transition system S, initial abstraction α, and property P

Example 7 Consider the fair transition system from Example 6 and the property $\mathbf{FG}\,\neg(c < d)$. K-liveness would reduce the check of $S \vDash \mathbf{FG}\,\neg(c < d)$ to find a bound K on the number of times S visits a state where $c < d$. Since the initial value of the variable d is unknown in the initial state, there is no upper bound on the value of K: after we fix a K, we can always pick a new path, choosing a smaller initial value for d, where the system visits a state where $c < d$ for $K + 1$ times.

4 CEGAR and predicate abstraction

Abstraction [51] is a technique used to reduce the search space while preserving the satisfaction of some properties. In symbolic model checking, the abstraction yields a simpler transition system $\widehat{S}$, possibly described by a different set of variables (denoted here by $\widehat{X}$). The abstraction is usually obtained by means of a surjective function $\alpha : \mathcal{S}_X \to \mathcal{S}_{\widehat{X}}$, called abstraction function, that maps the states of a symbolic fair transition system S into states of $\widehat{S}$. The concretization function $\gamma : \mathcal{S}_{\widehat{X}} \to 2^{\mathcal{S}_X}$ is defined as $\gamma(\widehat{s}) = \{s \in \mathcal{S}_X \mid \alpha(s) = \widehat{s}\}$. The abstraction function α is symbolically represented by a formula $H_\alpha(X, \widehat{X})$ such that $s, \widehat{s} \vDash H_\alpha$ iff $\alpha(s) = \widehat{s}$.

The *Counterexample Guided Abstraction Refinement (CEGAR) framework* [50] leverages abstraction to create a simplified version of the input transition system, which is amenable for finite state model checking. The abstraction is typically constructed to be conservative, that is, every trace in the concrete space has a counterpart in the abstract space. If there are no property violations in the abstract space, then there are no violations in the original system. However, if an abstract counterexample exists, there may not be a corresponding counterexample for the concrete system. Such an abstract counterexample is then called a *spurious counterexample*. Then, abstraction-refinement tries to discover a new abstract model, which contains more detail in order to rule out spurious counterexamples. This is done by extracting information from counterexamples generated by the model checker. The process is iterated until the property is either proved or disproved. In Fig. 1 we give a pictorial representation of the approach.

In the rest of this Section and in the following one, we focus on the instantiation of the CEGAR framework to *Predicate Abstraction* [52], while in Sect. 6 we focus on the instantiation of CEGAR to incremental linearization [11].

4.1 Computing the predicate abstraction

Predicate Abstraction [52] abstracts a transition system S with a transition system having as states the (finite) set of truth assignments to a set of predicates $\mathbb{P} = \{p_1(X), \ldots, p_m(X)\}$. A transition between two abstract states $\widehat{s_i}$ and $\widehat{s_j}$ in the abstraction is possible iff there exist two concrete states s_i and s_j such that the evaluation of the predicates in s_i is $\widehat{s_i}$, the evaluation in s_j is $\widehat{s_j}$, and $s_i, s_j' \models T$.

Predicate abstraction can be symbolically represented by associating to each predicate p a corresponding Boolean variable x_p. We define the *abstraction function* for predicate abstraction as:

$$H_{\mathbb{P}}(X, X_{\mathbb{P}}) := \bigwedge_{p \in \mathbb{P}} p(X) \leftrightarrow x_p. \tag{8}$$

The *abstract initial states*, the *abstract transition relation*, and the *abstract fairness condition* of the abstract transition system $\widehat{S} = \langle X_{\mathbb{P}}, \widehat{I}, \widehat{T}, \widehat{F} \rangle$ obtained by applying predicate abstraction to the concrete system are symbolically represented by the Boolean formulas $\widehat{I}(X_{\mathbb{P}})$, $\widehat{T}(X_{\mathbb{P}}, X'_{\mathbb{P}})$, and $\widehat{F}(X_{\mathbb{P}})$. The formulas are equivalent to the following definitions:

$$\widehat{I}(X_{\mathbb{P}}) := \exists X.(I(X) \wedge H_{\mathbb{P}}(X, X_{\mathbb{P}})), \tag{9}$$

$$\widehat{T}(X_{\mathbb{P}}, X'_{\mathbb{P}}) := \exists X, X'.(T(X, X') \wedge H_{\mathbb{P}}(X, X_{\mathbb{P}}) \wedge H_{\mathbb{P}}(X', X'_{\mathbb{P}})), \tag{10}$$

$$\widehat{F}(X_{\mathbb{P}}) := \exists X.(F(X) \wedge H_{\mathbb{P}}(X, X_{\mathbb{P}})). \tag{11}$$

The abstract fair transition system thus obtained is purely Boolean, and can be subjected to finite-state model checking to verify the abstraction of the properties of interest.

The computation of the abstraction (eqns. (9), (10) and (11)) is a key operation, and in the literature it has been addressed with several approaches. A first simple approach is based on encoding the problem in the quantified fragment of the theory $\mathcal{T}$, and on leveraging an SMT solver to perform the quantified elimination of the concrete variables X and X' to obtain the abstract counterpart. This has been done either by using specialized algorithms such as [53–55] or by leveraging general quantifier elimination procedures, e.g. [56–59]. All these approaches are subject to the model explosion problem. Indeed, they all end-up enumerating enough implicants to cover the abstraction, and this boils down to going through the construction of the corresponding DNF.

The works in [60, 61] tackle the problem of computing the abstraction by integrating BDD-based quantification techniques with SMT-based constraint solving. These approaches try to overcome the limitations of the previous approaches by exploiting the fact that BDDs are a DAG representation of the space that a SAT-based enumerator treats as a tree. In [61] the abstraction problem is no longer seen as a monolithic quantifier elimination problem, but a conjunctively-partitioned representation of the formula to quantify is leveraged to reduce the computation burden.

Finally, in [62] the idea of partitioning the computation, initially outlined in [61] was further expanded by providing a structure-aware abstraction algorithm. The proposed approach first exploits the high-level structure of the system, and partitions the abstraction problem into the combination of several smaller abstraction problems, still represented as a formula with quantifiers. Then, the low-level structure of the formula

(e.g. the occurrence of variables within the quantifiers, the application of low-level rewriting rules like De-Morgan and quantifier push) is leveraged to further reduce the scope of quantifiers. The resulting formulas (still with quantifiers) is then given in input to existing state-of-the-art quantifier elimination approaches like the ones in [56–61] to obtain a finite-state abstract model.

4.2 Refining the predicate abstraction from counterexamples

Let us now consider a sequence of abstract states $\widehat{s}_0, \cdots, \widehat{s}_k$ (where each abstract state $\widehat{s}_i$ is a valuation to the variables $X_{\mathbb{P}}$). If the abstract property does not hold in the abstract model, we generate an abstract counterexample that must be checked for spuriousness, i.e. we check whether it can be refined in the concrete space. This can be done with a setting similar to bounded model checking, where each state of both the concrete and abstract machine are replicated at different time steps, from 0 to k. Checking the spuriousness of a counterexample for an invariant property corresponds to checking if the following formula is unsatisfiable:

$$I(X^0) \wedge \bigwedge_{0 \leq h < k} T(X^h, X^{k+1}) \wedge \bigwedge_{0 \leq h \leq k} \left(H_{\mathbb{P}}(X^h, X^h_{\mathbb{P}}) \wedge \widehat{s}_h(X^h_{\mathbb{P}}) \right). \tag{12}$$

Similar considerations also hold in the case the property is a liveness one: if $\widehat{S}$ does not have an initial fair path, then the same can also be concluded for S. In this case, the check for the spuriousness of the counterexample shall also consider the fairness conditions F and encode a loop enforcing the fairness conditions to hold within the loop.

The main idea behind the refinement phase is to learn more information from the spurious counterexamples produced and use the information to refine the abstraction in such a way that it rules out the spurious counterexample. Spurious transitions are those abstract transitions that do not have any corresponding concrete transitions. If the most precise abstraction with respect to the given set of predicates is computed, the spuriousness of the counterexample would be because of an insufficient number of predicates, i.e. the absence of information rich enough to capture all the relevant behaviors of the concrete system, even for the most precise abstraction. Several approaches have been proposed in the literature to extract new predicates in the refinement. Most of them are based on the analysis of the unsatisfiable cores or the computation of Craig interpolants of the spuriousness check formula (12) (e.g., [63, 64]).

5 Invariant checking with implicit predicate abstraction

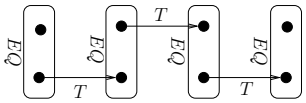
5.1 Implicit predicate abstraction

As also defined in [51, 65], the abstraction induces an equivalence relation among the concrete states:

$$s_1 \sim s_2 \Leftrightarrow \alpha(s_1) = \alpha(s_2),$$

which in the case of predicate abstraction is characterized by the following formula:

Fig. 2 Abstract path



$$EQ_{\mathbb{P}}(X, \overline{X}) := \bigwedge_{p \in \mathbb{P}} p(X) \leftrightarrow p(\overline{X}). \tag{13}$$

The formula $EQ_{\mathbb{P}}$ asserts that two concrete states have a consistent evaluation of predicates. It can be exploited to embed the abstraction into formulas over the concrete variables, thus reasoning about abstract states without explicitly computing them. This idea, first proposed [9] and later expanded in [10], takes the name of Implicit Abstraction (IA).

The clear advantage of IA is that it avoids the computation of the abstract transitions that must be done upfront in the explicit predicate abstraction and which results typically in a bottleneck. In fact, in the explicit abstraction case, the model checking algorithm on the abstract transition system cannot start before computing abstract transition (this is partly alleviated in Lazy Abstraction [66] where the abstraction is localized and computed on the fly based on an explicit-state control graph). On the other side, IA reasons over the concrete variables and thus, in some cases, cannot exploit simplification available only in the propositional case.

More in detail, IA embeds the definition of the predicate abstraction in the encoding of a path. The formula $Path^k_{\mathbb{P}} := \bigwedge_{1 \le h < k} (T(\overline{X}^{h-1}, X^h) \wedge EQ_{\mathbb{P}}(X^h, \overline{X}^h)) \wedge T(\overline{X}^{k-1}, X^k)$ is satisfiable iff there exists a path of k steps in the abstract state space. 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).

We can therefore extend $Path^k_{\mathbb{P}}$ to solve an abstract bounded model checking problem. Suppose we want to verify that a property $P(X)$ holds in the abstract state space in all abstract states reachable in k abstract steps. We can encode the dual problem into a formula $BMC^k_{\mathbb{P}}$ that uses $Path^k_{\mathbb{P}}$ to assert that there exists an abstract path, the first state is initial, and the last state satisfies the abstraction of $\neg P$. We use again $EQ_{\mathbb{P}}$ to build this encoding just using concrete variables as follows:

$$BMC^k_{\mathbb{P}} = I(X^0) \wedge EQ_{\mathbb{P}}(X^0, \overline{X}^0) \wedge Path^k_{\mathbb{P}} \wedge EQ_{\mathbb{P}}(X^k, \overline{X}^k) \wedge \neg P(\overline{X}^k).$$

5.2 K-induction with implicit predicate abstraction

Similarly to the encoding of bounded model checking, we can define the abstract version of the k-induction conditions with IA:

$$KINDFW^k_{\mathbb{P}} := I(\overline{X}_0) \wedge EQ_{\mathbb{P}}(\overline{X}_0, X_0) \wedge SIMPLE^k_{\mathbb{P}}, \tag{14}$$

$$KINDBW^k_{\mathbb{P}} := SIMPLE^k_{\mathbb{P}} \wedge EQ_{\mathbb{P}}(X_k, \overline{X}_k) \wedge \neg P(\overline{X}_k), \tag{15}$$

where

$$SIMPLE^k_{\mathbb{P}} := Path^k_{\mathbb{P}} \wedge \bigwedge_{0 \le i < j \le k} \neg EQ_{\mathbb{P}}(X_i, X_j). \tag{16}$$

Therefore, if $BMC^k_{\mathbb{P}}$ is unsatand, either $KINDFW^k_{\mathbb{P}}$ or $KINDBW^k_{\mathbb{P}}$ is unsat, then we can conclude that the invariant is not reachable in the abstract state space (and thus either in the concrete system).

Notice that we do not use the stronger version of $KINDFW^k_{\mathbb{P}}$ and $KINDBW^k_{\mathbb{P}}$ defined in [5], because they require to express the negation of the abstraction of P, which involves an existential quantification, and their negation cannot be handled by the satisfiability solver.

5.3 IC3IA: IC3 with implicit abstraction

Implicit Abstraction provides a simple, yet very effective, way of generalising IC3 from SAT to SMT. The main idea is that of making IC3 work on the abstract state space defined by a set of predicates $\mathbb{P}$, and use implicit abstraction to avoid the explicit computation of the abstract transition relation. In the modified algorithm, which we call IC3IA, clauses, frames and cubes are formulas over the set $X_{\mathbb{P}}$ of abstract variables. When working in the abstract space, the critical step for IC3IA is repeatedly checking whether a clause c is inductive relative to the frame $\mathcal{F}$ (where c and $\mathcal{F}$ are both formulas over $X_{\mathbb{P}}$). This check, if encoded as $RelInd(\mathcal{F}, \widehat{T}, c)$, would require the explicit construction of $\widehat{T}$. The key insight underlying IC3IA is to use implicit abstraction to perform the check without actually constructing the abstract transition relation $\widehat{T}$. This is done by checking the quantifier-free formula:

$$\begin{aligned} AbsRelInd(\mathcal{F}, T, c, \mathbb{P}) := {} & \mathcal{F}(X_{\mathbb{P}}) \wedge c(X_{\mathbb{P}}) \wedge H_{\mathbb{P}}(X, X_{\mathbb{P}}) \wedge H_{\mathbb{P}}(X', X'_{\mathbb{P}}) \wedge \\ & EQ_{\mathbb{P}}(X, \overline{X}) \wedge T(\overline{X}, \overline{X}') \wedge EQ_{\mathbb{P}}(\overline{X}', X') \wedge \neg c(X'_{\mathbb{P}}). \end{aligned} \tag{17}$$

It can be shown (see [67]) that working with *AbsRelInd* is equivalent to working with *RelInd* on the abstract transition relation $\widehat{T}$, in the sense that every model μ for $AbsRelInd(\mathcal{F}, T, c, \mathbb{P})$ is also a model for $RelInd(\mathcal{F}, \widehat{T}, c)$ (when appropriately projected to $X_{\mathbb{P}} \cup X_{\mathbb{P}'}$). The consequence of this is that we can obtain an SMT-aware version of IC3 operating on a predicate abstraction of the original system by simply replacing the underlying SAT solver with an SMT solver, and using *AbsRelInd* instead of *RelInd* for performing the relative induction checks. Thanks to implicit predicate abstraction, we therefore obtain an algorithm that is simple, flexible (automatically supporting all theories that are handled by the underlying SMT solver), and very competitive in practice [67].

Example 8 Take the transition system and property from Example 4 and the set of predicates $\mathbb{P} = \{(c = 0), (d = 0), (d \le 3), (c \le d)\}$. The algorithm eventually checks that $AbsRelInd(\mathcal{F}_0, T, c_0, \mathbb{P})$, where $\mathcal{F}_0 := \{x_{c=0}, x_{d=0}\}$, $c_0 := x_{c=0} \wedge \neg x_{d=0} \wedge \neg x_{d\le 3} \wedge x_{c\le d}$, and $x_p \in X_{\mathbb{P}}$ denotes the abstract variable for the predicate $p \in \mathbb{P}$ (e.g., $x_{d=0}$ is the abstract variable of the predicate $d = 0$):

$$
\begin{aligned}
&\mathit{AbsRelInd}(F_0, T, c_0, \mathbb{P}_0) \ := \\
&\quad x_{c=0} \wedge x_{d=0} \wedge && [F_0(X_\mathbb{P})] \\
&\quad x_{c=0} \wedge \neg x_{d=0} \wedge \neg x_{d\leq 3} \wedge x_{c\leq d} \wedge && [c_0(X_\mathbb{P})] \\
&\quad x_{d=0} \leftrightarrow (d=0) \wedge x_{c=0} \leftrightarrow (c=0) \wedge && [H_\mathbb{P}(X, X_\mathbb{P})] \\
&\quad x_{d\leq 3} \leftrightarrow (d \leq 3) \wedge x_{c\leq d} \leftrightarrow (c \leq d) \wedge \\
&\quad x'_{d=0} \leftrightarrow (d'=0) \wedge x'_{c=0} \leftrightarrow (c'=0) \wedge && [H_\mathbb{P}(X', X'_\mathbb{P})] \\
&\quad x'_{d\leq 3} \leftrightarrow (d' \leq 3) \wedge x'_{c\leq d} \leftrightarrow (c' \leq d') \wedge \\
&\quad (d=0) \leftrightarrow (\overline{d}=0) \wedge (c=0) \leftrightarrow (\overline{c}=0) \wedge && [EQ_\mathbb{P}(X, \overline{X})] \\
&\quad (d \leq 3) \leftrightarrow (\overline{d} \leq 3) \wedge (c \leq d) \leftrightarrow (\overline{c} \leq \overline{d}) \wedge \\
&\quad (\overline{c'} = \overline{c} + \overline{d}) \wedge (\overline{d'} = \overline{d} + 1) \wedge && [T(\overline{X}, \overline{X}')] \\
&\quad (\overline{d'}=0) \leftrightarrow (d'=0) \wedge (\overline{c'}=0) \leftrightarrow (c'=0) \wedge \\
&\quad (\overline{d'} \leq 3) \leftrightarrow (d' \leq 3) \wedge (\overline{c'} \leq \overline{d'}) \leftrightarrow (c' \leq d') \wedge && [EQ_\mathbb{P}(\overline{X}', X')] \\
&\quad \neg(x'_{c=0} \wedge \neg x'_{d=0} \wedge \neg x'_{d\leq 3} \wedge x'_{c\leq d}) && [\neg c_0(X'_\mathbb{P})]
\end{aligned}
$$

Since $\mathit{AbsRelInd}(\mathcal{F}_0, T, c_0, \mathbb{P}) \vDash \bot$, it's also the case case that $\mathit{RelInd}(\mathcal{F}_0, \widehat{T}, c) \vDash \bot$ in the abstract system (see [67] for a full example of a run of IC3IA).

5.4 Refining implicit predicate abstractions

Like all techniques based on predicate abstraction, when IC3IA finds an abstract counterexample $\widehat{\pi} := \widehat{s_0}, \widehat{s_1}, \ldots, \widehat{s_k}$ (in the form of an interpretation of $\bigcup_{i=0}^{k} X^i_\mathbb{P}$), it must check whether it can be concretized, i.e. whether there exists a corresponding counterexample in S. This is done by simulating the abstract counterexample $\widehat{\pi}$ in the concrete system S, by encoding all the paths of S up to k steps restricted to $\widehat{\pi}$:

$$
\bigwedge_{0\leq i<k} T(X^i, X^{i+1}) \wedge \bigwedge_{0<i\leq k} \widehat{s_i}(X^i_\mathbb{P})[\mathbb{P}(X^i)/X^i_\mathbb{P}], \tag{18}
$$

where $\widehat{s_i}(X^i_\mathbb{P})[\mathbb{P}(X^i)/X^i_\mathbb{P}]$ is the formula obtained from $\widehat{s_i}(X^i_\mathbb{P})$, seen as a conjunctions of literals, by replacing each Boolean variable in $X^i_\mathbb{P}$ by the corresponding predicate over X^i.

If (18) is satisfiable, then the interpretation of the concrete variables $X^0, \ldots, X^k$ yields a concrete counterexample $s_0, s_1, \ldots, s_k$ witnessing the violation of P. Otherwise, $\widehat{\pi}$ is spurious, and the abstraction must be refined by adding new predicates. The refinement procedure is somewhat orthogonal to IC3IA, and can be done in various ways [66, 68, 69]. The only requirement is that the new set of predicates should be sufficient to remove the spurious counterexample. A popular approach is to use SMT-based interpolation to discover new predicates, as described in [68]. Although this technique always returns a set of predicates that are sufficient to refute the spurious counterexample, it offers no guarantee that all the discovered predicates are necessary. In other words, predicate discovery via interpolation can produce redundant predicates, which cause an increase in the precision of the predicate abstraction which might be not necessary, thus potentially slowing down the convergence of IC3IA. This drawback can however be mitigated by exploiting implicit abstraction also for detecting redundant predicates. Let $\mathbb{P}_{\text{new}}$ be the set of predicates produced by the refinement procedure, such that the set $\mathbb{P} \cup \mathbb{P}_{\text{new}}$ is sufficient to refute

the abstract counterexample. By definition, this means that $\widehat{\pi}$ is not a path of the abstract system $\widehat{S}$, i.e. the predicate abstraction of S, wrt. $\mathbb{P} \cup \mathbb{P}_{\text{new}}$. Formulating this in terms of implicit abstraction means that the formula

$$\begin{aligned}\bigwedge_{0<i\leq k} \widehat{s}_i(X^i_{\mathbb{P}}) \bigwedge_{0<i<k} \Big(& T(\overline{X}^i, \overline{X}^{i+1}) \wedge \\ & H_{\mathbb{P}}(X^i, X^i_{\mathbb{P}}) \wedge H_{\mathbb{P}}(X^{i+1}, X^{i+1}_{\mathbb{P}}) \wedge \\ & EQ_{\mathbb{P}}(X^i, \overline{X}^i) \wedge EQ_{\mathbb{P}}(\overline{X}^{i+1}, X^{i+1}) \wedge \\ & H_{\mathbb{P}_{\text{new}}}(X^i, X^i_{\mathbb{P}_{\text{new}}}) \wedge H_{\mathbb{P}_{\text{new}}}(X^{i+1}, X^{i+1}_{\mathbb{P}_{\text{new}}}) \wedge \\ & EQ_{\mathbb{P}_{\text{new}}}(X^i, \overline{X}^i) \wedge EQ_{\mathbb{P}_{\text{new}}}(\overline{X}^{i+1}, X^{i+1}) \Big)\end{aligned} \tag{19}$$

is unsatisfiable. We can use this fact to check for predicate redundancy, e.g. by heuristically replacing $\mathbb{P}_{\text{new}}$ with one of its subsets as long as (19) is still unsatisfiable. In practice, this can be performed efficiently by exploiting the capability offered by modern SMT solvers of (i) solving a formula under a set of assumptions, and (ii) producing an unsatisfiable core of the assumptions in case the formula is not satisfiable. More in detail, for each predicate $p \in \mathbb{P}_{\text{new}}$, we can generate a fresh label variable l_p. Then, we replace each formula $EQ_{\mathbb{P}_{\text{new}}}(X^i, \overline{X}^i)$ in (19) with $\bigwedge_{p\in\mathbb{P}_{\text{new}}} l_p \rightarrow (p(X^i) \leftrightarrow p(\overline{X}^i))$ (and similarly for $EQ_{\mathbb{P}_{\text{new}}}(\overline{X}^{i+1}, X^{i+1})$). Finally, we solve under the assumptions $\{l_p \mid p \in \mathbb{P}_{\text{new}}\}$, and mark all the predicates p for which l_p is not in the unsat core as redundant.

6 CEGAR via incremental linearization

Predicate abstraction is not the only approach to implement a CEGAR loop. In fact, the techniques presented in the previous sections are not always applicable, since they rely on several strong assumptions on the ability of the SMT solver: first, the SMT solver is given a large number of satisfiable queries; second, the SMT solver must expose an incremental interface; third, it must be able to provide interpolation and quantifier elimination. These requirements become hard to satisfy when dealing with nonlinear theories, that allow for multiplications between real- or integer-valued variables, or for transcendental functions such as exponentiation and trigonometric functions. Hence, despite the power of theory solvers based on Cylindrical Algebraic Decomposition [70] and effective implementations like Z3 [71], the direct integration of a nonlinear SMT solver inside the IC3IA algorithm would not be practical.

We describe a CEGAR loop for the theory of nonlinear arithmetic with transcendental functions ($\mathcal{NTA}$). The approach, that is not based on predicate abstraction, is called *Incremental Linearization* [11] (Fig. 3). The idea is to abstract the VMT($\mathcal{NTA}$) problem in the combined theory of linear real arithmetic and the theory of equality with uninterpreted functions ($\mathcal{UFLRA}$). Specifically, nonlinear multiplications and transcendental functions are abstracted as *uninterpreted* function symbols. For example, the formula

$$x * x + y * y \leq 2 \wedge (x \geq 1.1 \vee x \leq -1.1) \wedge (x + 3 * y \geq 1.1 \vee 2 * \sin(x) \leq -1.1)$$

is abstracted to

$$f_*(x, x) + f_*(y, y) \leq 2 \wedge (x \geq 1.1 \vee x \leq -1.1) \wedge (x + 3 * y \geq 1.1 \vee 2 * f_{\sin}(x) \leq -1.1)$$

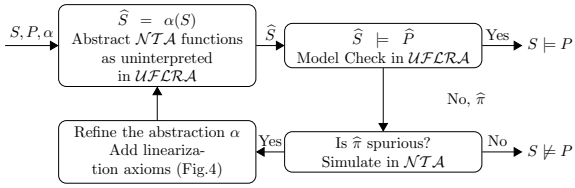


Fig. 3 The VMT($\mathcal{NTA}$) CEGAR loop via Incremental Linearization

We notice that the linear multiplications (e.g. $3 * y$) are not abstracted, while the nonlinear applications of $*$ and sin are replaced by the uninterpreted functions $f_*(\cdot)$ and $f_{\sin}(\cdot)$. This abstraction is clearly conservative. Hence, if the property holds in the abstract space, then the concrete system can be deemed to be safe. However, if a counterexample can be found in the abstract space, a concretization step in VMT($\mathcal{NTA}$) is required. If the abstract counterexample can be concretized, then the property does not hold. Otherwise, it is necessary to refine the abstraction by restricting the interpretation of the uninterpreted symbols in VMT($\mathcal{UFLRA}$).

Interestingly, the approach builds upon a black-box invariant checker for VMT($\mathcal{UFLRA}$), which could be based on a complex abstraction refinement loop as described in the previous sections. Precise reasoning in $\mathcal{NTA}$is limited to SMT in the concretization phase, given that the reason for spuriousness is that some uninterpreted functions may have been misinterpreted. For example, it is possible that the interpretation of the abstraction of multiplication of x and y, referred to as $\widehat{\mu}[f_*(x, y)]$, does not respect the semantics of multiplication, i.e. $\widehat{\mu}[f_*(x, y)] \neq \widehat{\mu}[x] * \widehat{\mu}[u]$. In order to rule out such spurious models, various patterns of linear axioms are introduced (see Fig. 4). These include some basic facts (e.g. sign rules, commutativity, multiplication by zero), monotonicity, and *tangent plane approximation*. The latter dynamically constrains the multiplication on point (a, b), when the interpretation of $f_*(x, y)$ is such that $\widehat{\mu}[f_*(x, y)] \neq a * b$, with $a = \widehat{\mu}[x]$ and $b = \widehat{\mu}[y]$. The idea is to approximate the multiplication function, that is a hyperbolic paraboloid (Fig. 5, upper left), with a tangent plane centered around (a, b). In addition to constraining the value on the specific point, we can see that the plane intersects the multiplication curve on straight lines expressible in the form of linear equalities (Fig. 5, lower left), and in the resulting quadrants it can be used to express upper- and lower-bounds.

Additional attention is required to deal with approximations of nonlinear transcendental functions. First, since irrational values are not directly representable, one needs to make sure that the piece-wise linear functions are correct (over- or under-) approximations, which depends on the actual concavity of the curve. Second, trigonometric functions are dealt with by leveraging periodicity, reducing reasoning to the base period between $-\pi$ and π and ensuring that all the lemmas can be applied to the other periods. We refer to [11] for the details.

Despite its simplicity, incremental linearization leads to significant results for VMT($\mathcal{NTA}$), both in terms of expressiveness as well as in terms of effectiveness. The approach is motivated by the fact that many practical applications are "mostly linear", in the sense that only a small percentage of the constraints are nonlinear. Hence, the idea is to look for a piece-wise linear invariant that is strong enough to prove the property, so that expensive nonlinear reasoning is replaced – as much as possible – by cheaper linear reasoning.

Finally, we note that the idea of incremental linearization not only applies to VMT but also to SMT($\mathcal{NTA}$). In fact, incremental linearization is akin to a lemmas-on-demand approach,

Fig. 4 Axioms for nonlinear multiplication refinement

- $(abs(x_1) \leq abs(x_2) \wedge abs(y_1) \leq abs(y_2)) \rightarrow abs(f_*(x_1, y_1)) \leq abs(f_*(x_2, y_2)))$
- $(abs(x_1) < abs(x_2) \wedge abs(y_1) \leq abs(y_2) \wedge y_2 \neq 0) \rightarrow abs(f_*(x_1, y_1)) < abs(f_*(x_2, y_2)))$
- $(abs(x_1) \leq abs(x_2) \wedge abs(y_1) < abs(y_2) \wedge x_2 \neq 0) \rightarrow abs(f_*(x_1, y_1)) < abs(f_*(x_2, y_2)))$
- $f_*(a, y) = a * y \wedge f_*(x, b) = b * x \wedge$
  $((x > a \wedge y < b) \rightarrow f_*(x, y) < \textsc{TanPlane}_{*,a,b}(x, y)) \wedge$
  $((x < a \wedge y > b) \rightarrow f_*(x, y) < \textsc{TanPlane}_{*,a,b}(x, y)) \wedge$
  $((x < a \wedge y < b) \rightarrow f_*(x, y) > \textsc{TanPlane}_{*,a,b}(x, y)) \wedge$
  $((x > a \wedge y > b) \rightarrow f_*(x, y) > \textsc{TanPlane}_{*,a,b}(x, y))$
  where $a = \widehat{\mu}[x]$ and $b = \widehat{\mu}[y]$

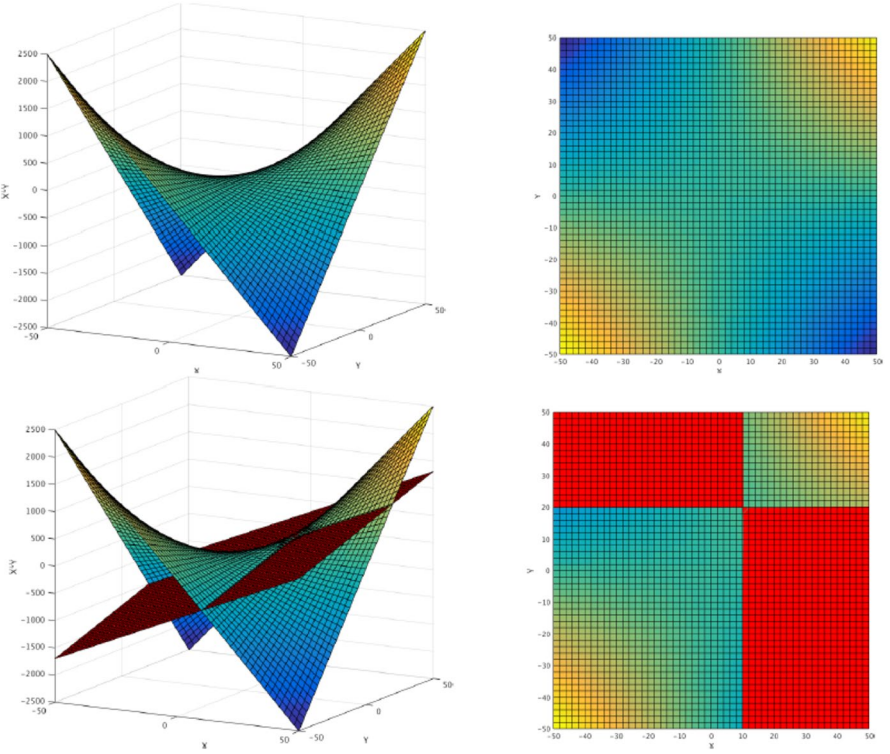


Fig. 5 A graphical view of tangent plane approximation

where the interpretation of nonlinear multiplications and transcendental functions is progressively restricted by the introduction of lemmas that will rule out spurious abstract models (in the case of SMT) or counterexamples (in VMT).

7 LTL(T) model checking

7.1 Liveness to safety via implicit abstraction and well-founded relations

As described in the previous sections, various approaches to LTL Model Checking reduce the problem to the verification of an invariant. Such reductions may be neither complete nor correct in the case of infinite-state systems. Some of the issues are that, on one hand, if there exists a counterexample, it is not guaranteed that there exists a lasso-shaped one; on the other hand, if there is no counterexample visiting a live signal infinitely many times, there may be no bound on the number of such visits.

In order to cope with these issues, in [12], we proposed an approach integrating liveness-to-safety with implicit abstraction and well-founded relations. By applying implicit abstraction to the liveness-to-safety encoding, we can effectively prove the absence of abstract fair loops without explicitly constructing the abstract state space. The approach is extended by using termination techniques based on well-founded relations derived from ranking functions: the idea is to prove that any existing abstract fair loop is covered by a given set of well-founded relations. Within this framework, k-liveness is integrated as a generic ranking function. The algorithm iterates by attempting to remove spurious abstract fair loops: either it finds new predicates, to avoid spurious abstract prefixes, or it introduces new well-founded relations, based on the analysis of the abstract lasso. The implementation fully leverages the efficiency and incrementality of the underlying safety checker IC3IA.

More specifically, after encoding the LTL model checking problem into a liveness problem in the form $\mathbf{FG}\neg f$ on a transition system S, we produce a sequence of invariant checking problems $S_0 \vDash_{inv} \phi_0$, $S_1 \vDash_{inv} \phi_1$, $\cdots$. For each j, S_j and ϕ_j are the result of an encoding operation dependent on given sets of *state predicates* P and *well-founded relations* W: S_j, $\phi_j = \text{ENCODE}(S, f, P, W)$. ENCODE is a variant of the liveness-to-safety transformation that includes both implicit predicate abstraction and well-founded relations: if $S_j \vDash_{inv} \phi_j$ either there is no abstract fair loop or every such loop is covered by the given set of well-founded relations. Thus, it ensures that if $S_j \vDash_{inv} \phi_j$, then $S \vDash \mathbf{FG}\neg f$, in which case the iteration terminates. If $S_j \nvDash_{inv} \phi_j$, we analyze a (finite) counterexample trace π in S_j to determine whether it corresponds to an (infinite) counterexample for $\mathbf{FG}\neg f$ in S. If so, then we conclude that the property doesn't hold. Otherwise, if we can conclude that π doesn't correspond to any real counterexample in S, we try to extract new predicates P' and/or well-founded relations W' to produce a refined encoding: $S_{j+1}, \phi_{j+1} := \text{ENCODE}\,(S, f, P \cup P', W \cup W')$, where $P', W' := \text{ENCODE}\,(S_j, \pi, P, W)$. If we can neither confirm nor refute the existence of real counterexamples, we abort the execution, returning "unknown".[3]

Example 9 Consider the transition system $S = \langle \{c, d\}, c = 0 \wedge d \geq 0, c' = c + 1 \wedge d' = d, \top \rangle$, the LTL($\mathcal{T}$) property $\mathbf{FG}\, c > d$, and the initial set of predicates $\mathbb{P} := \{c \leq d, c = 0, 0 \leq d\}$. The algorithm will first find an abstract lasso-shaped counter-example with prefix $\{x_{c\leq d}, x_{c=0}, x_{0\leq d}\}$ and a self loop on the abstract state $\{x_{c\leq d}, \neg x_{c=0}, x_{0\leq d}\}$. The algorithm cannot determine that such abstract counter-example is spurious using bounded model checking, since there is no corresponding lasso-shaped path in S. Instead, the algorithm synthesizes a ranking function $d - c$, with lower bound $-1 \leq d - c$, proving the the abstract loop

[3] We might also diverge and/or exhaust resources in various intermediate steps (e.g. in checking $S_j \vDash \phi_j$ or during refinement).

terminates. The algorithm uses the ranking function to get a well-founded relation. We refer to [12] for the details of the encoding of the ranking function in the liveness-to-safety reduction: intuitively, the encoding relaxes the *loop* condition used in L2S (see Sect. 3.4.2), so that abstract loops that satisfy at least a well founded relation are not considered as violation for the LTL property.

7.2 The K-Zeno algorithm

The K-liveness algorithm is sound and complete for finite-state systems (i.e., the implication in (7) holds also in the other direction) although, in practice, the algorithm is effective only for proving liveness, rather than finding a violation. The K-liveness algorithm is *sound*, however not complete, to prove liveness properties for infinite-state systems. In an infinite-state system the bound K may not be a specific natural number but instead may depend on the system variables (the bound K is a function of the state variables of the transition system S). For example, consider an infinite-state system S with an integer parameter $p \geq 0$ where the condition f can be visited at most p times. In such system the property $S \vDash \mathbf{FG}\neg f$ holds, but K-liveness would never prove that since for all $K \in \mathbb{N}$, there exists a value of $p \geq 0$ such that $S \vDash p > K$. In practice, once we fix the value of K to an natural number to check $S \vDash \#(f) \leq K$, the path where $p = K + 1$ is sufficient to show that $S \nvDash \#(f) \leq K$.

The above limitation of K-liveness also affects the analysis of infinite-state systems with paths where the value of a variable diverges. We obtain infinite-state transition systems with such diverging paths when we encode timed automata [47] and hybrid automata [72],[4] two formalisms that model respectively real-time systems and control systems. Both formalisms do not exclude *fair Zeno paths*, infinite paths of the system where the real time does not diverge. We call *non-Zeno* paths all the infinite paths where time diverges. However, the fairness verification problem for timed and hybrid automata does not consider such Zeno paths, since they unrealistically assume that an infinite number of computation steps can happen in a finite amount of time [72]. K-liveness would not prove liveness properties when naïvely applied to the transition system S encoding of a timed or hybrid automaton that contains Zeno paths where the fairness condition f holds. In fact, for every possible value of K, the transition system would have an infinite path where the real time variable does not diverge and where the the condition f holds. This means that, for any choice of K, we would have that $S \nvDash \#(f) \leq K$.

The K-Zeno [13] algorithm applies a transformation to the transition system S to consider only occurrences of the fairness condition f on paths where time diverges. In such transformation the algorithm uses a new transition system S_β that introduces a *bound* β on the real time elapsed between two consecutive occurrences of the fairness condition f. In this way, the system only considers occurrences of the condition f that happen on a non-Zeno path. The liveness verification problem that considers only non-Zeno paths where f may hold is then:

$$\exists K \cdot S \times S_\beta \vDash \#(f) \leq K \tag{20}$$

[4] See [73–76] for details about possible encoding of timed and hybrid systems as infinite-state transition systems.

The specific construction of the transition system S_β depends on the kind of system to verify. For example, if the system is a timed automaton (without parameters), the bound β can be just a constant (the maximum constant in the model or 1). The transition system S_β may have to capture a symbolic bound $\beta(X)$ that is a function of the state variables X, instead of being just a constant, when analyzing more general classes of systems (e.g., hybrid automata). The work in [13] shows that there exists a monitor $S_{\beta(X)}$ that guarantees a complete reduction in the case of initialized with bounded non-determinism parametric hybrid automata (i.e., if $S \models \mathbf{FG}\neg f$ then $\exists K.S \times S_{\beta(X)} \models \#(f) \leq K$).

7.3 Beyond lasso-shaped counterexamples

In finite-state systems, if an LTL property is false, there is always a counterexample path (i.e. a witness) for it which is ultimately periodic (i.e. in a lasso-shaped form). When dealing with the infinite-state case, this is no longer the case, as in general in an infinite-state system a false LTL property might admit no lasso-shaped witness. Therefore, in order to effectively find counterexamples for LTL properties in infinite-state systems, it is necessary to look for ways to encode a more general class of infinite traces.

This problem has been investigated extensively in the context of software (non)termination. In that setting, closed recurrence sets [77] are used to represent a witness for the nontermination of some software program. A closed recurrence set consists of a reachable set of states that is disjoint from the end states and inductive with respect to a left-total transition relation that underapproximates the transition relation of the program. The set represents at least one infinite execution for the program: (i) its reachability ensures that there is some finite execution of the program ending in some state within the set; (ii) since the set is also inductive, we know that no transition starting from within the set can reach a state outside of it and (iii) the left-total transition relation ensures that there always exists at least one successor state satisfying also the transition relation of the program.

However, in the more general context of counterexamples for full LTL properties, recurrent sets are not sufficient, as a counterexample trace has the additional requirement of being *fair*, i.e. it needs to visit some fair state infinitely often. Unless the set underapproximates the fair states, without additional information, we cannot conclude that the infinite executions described by the closed recurrence set are fair.

In order to solve this problem, we have recently generalised the notion of recurrence set to take fairness conditions into proper account [78]. We split the closed recurrence set into two components S and D, such that D is a subset of the fair states. The union of S and D must satisfy the same conditions described above for closed recurrence sets and, in addition, the left-total transition relation must not allow for infinite sequences of S states: every state in S must reach a state in D in a finite number of steps. Moreover, in order to aid the automatic discovery of such generalised recurrence sets, we split the monolithic problem described above into two orthogonal directions: by *segmenting* the infinite paths into finite paths and *decomposing* the system with respect to some partitioning of the symbols.

More precisely, we segment the fair paths into a concatenation of finite paths: we split S into multiple regions such that each region represents a set of finite paths that must eventually reach the following region. Notice that, while each path in a region must be finite, there might be no upper bound to their length: a region can represent an infinite number of finite paths with increasing lengths. We call each segment *funnel* and their concatenation representing the fair paths *funnel-loop*. In addition, we *decompose* the system by partitioning its symbols. Each component, called E-component (for existential component), describes

the behavior of a subset of the symbols while assuming some properties about the others. These properties represent the conditions that are necessary for this behavior to be enabled and we need to prove that such conditions are ensured by some other component.

With this partitioned representation, which we have proven to be both sound and relatively complete, we have then developed a search procedure, based on a combination of invariant checking with abstraction, bounded model checking and SMT-based synthesis via quantified reasoning, that is capable of identifying funnel-loops in an automatic manner [78], with an implementation that significantly outperforms the state of the art on a wide class of benchmarks.

Example 10 Consider the fair transition system:

$$\langle \{c, n, o\}, \top, (n > o) \wedge ((c < n \wedge c' = c + 1 \wedge n' = n \wedge o' = o) \vee (c \leq n \wedge o' = n)), c = 0 \rangle.$$

The system admits fair paths that are not lasso-shaped. For example, one such path is the following (where the states show the values for the variables c, n, o, and k is an integer constant):

$$\{0,1,0\}, \{1,1,0\}, \{0,2,1\}, \{1,2,1\}, \ldots, \{k-1,k,k-1\}, \{k,k,k-1\}, \{0,k+1,k\}, \ldots$$

We can represent the paths above using a funnel as defined in [78], which intuitively defines an underapproximation of the original system that contains only fair paths. The funnel is defined as having a source region $S := (o > 0 \wedge n > o)$ (which is a subset of the initial states of the system), a destination region $D := (o > 0 \wedge o < n \wedge c = 0)$ (which is a subset of S consisting of fair states), connected by a transition relation $T := (c < n \wedge c' = c + 1 \wedge n' = n \wedge o' = o) \vee (c \geq n \wedge o' = n \wedge c' = 0 \wedge n' = n + 1)$, which is an underapproximation of the transition relation of the system ensuring that eventually D must be reached when starting from S.[5]

8 The nuXmv model checker

All the techniques and algorithms described in the previous sections have been implemented in nuXmv [16], a state-of-the-art symbolic model checker for both finite- and infinite-state synchronous fair transition systems. nuXmv is the successor of NuSMV [17], the popular symbolic model checker for finite-state systems which was conceived in 1999 within a joint cooperation of the CMU group of Ed Clarke and FBK.[6] In some sense, therefore, nuXmv carries on the legacy of Ed Clarke and his message of combining theoretical results with strong practical applications.

From a technical standpoint, nuXmv integrates, extends and complements the functionalities of NuSMV with multiple functionalities to facilitate its deployment in several operational industrial and research settings. For finite-state systems, it complements the basic verification techniques available in NuSMV with a family of new state-of-the-art SAT-based techniques, including interpolation and IC3. For infinite-state systems,

[5] The details are omitted for simplicity. We refer to [78] for a full definition of funnels and their properties.

[6] At that time Istituto Trentino di Cultura.

Fig. 6 nuXmv listing for the transition system of Example 4

```
MODULE main

INVARSPEC (d <= 3) | ( ! (c <= d));

LTLSPEC F G c >= d;

VAR
  c : integer;
  d : integer;

INIT
  c = 0 & d = 0;

TRANS
  next(c) = c + d & next(d) = d + 1;
```

nuXmv extends the NuSMV language with new data types, namely integers, reals and unbounded arrays. Figure 6 shows the nuXmv specification for the transition system and property from Example 4. The example shows how the nuXmv language can express infinite state transition systems, specifying the variables (VAR declarations), the initial condition (INIT declaration), and the transition relation (TRANS) declaration. Notice that the nuXmv input language is way richer, allowing to specify, for example, modules, fairness constraints, and timed transition systems [79]. nuXmv provides advanced SMT-based model checking techniques and implements all the abstraction-based approaches discussed in the previous sections (including several explicit computation techniques, e.g., [60–62]). Moreover, it complements these algorithms with other functionalities whose description is out of scope of this paper. The interested reader can refer to [16] for detailed discussion of all the functionalities provided by nuXmv.

nuXmv has been used in a wide range of applications, both at academic and at industrial level, in different application domains including avionics, railways, automotive, space, and biological (e.g. [24, 31, 32, 80–84].) Finally, nuXmv is also the back-end of several other tools, including the Kratos [85] software model checker, the RATSY [86] tool for temporal logic synthesis, the OCRA [22] platform for contract-based verification, the xSAP [87] for model-based safety assessment, and the HyCOMP [20] model checker for the verification of hybrid systems.

9 Conclusions

In this paper we presented a retrospective on the verification of infinite-state transition systems expressed symbolically in SMT. This line of work was substantially influenced by the work of Ed Clarke, based on the ideas on SAT-based model checking and abstraction refinement. At the methodological level, Clarke always underlined the importance of developing strong tools and applying them in practical case studies, being enthusiastic about the work on the NuSMV model checker. This paved the way to the development of the nuXmv model checker, and its applications to practical industrial case studies.

Challenges for future research include devising effective verification and falsification algorithms and tools for more expressive classes of VMT problems: parameterized systems expressed in the quantified theory of arrays [88], timed and hybrid systems [40], sequential and concurrent software, recurrent neural networks, and the closed-loop combination of physical systems and control software.

Acknowledgements We acknowledge the support of the PNRR project FAIR - Future AI Research (PE00000013), under the NRRP MUR program funded by the Next Generation EU.

Availability of data and materials The manuscript has no associated data.

References

1. Clarke EM, Emerson EA (1981) Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Kozen D (ed) Logics of programs, workshop, yorktown heights, New York, USA, May 1981. Lecture Notes in Computer Science, vol 131, pp 52–71. https://doi.org/10.1007/BFb0025774. https://doi.org/10.1007/BFb0025774
2. Queille J, Sifakis J (1982) Specification and verification of concurrent systems in CESAR. In: Dezani-Ciancaglini M, Montanari U (eds) International symposium on programming, 5th colloquium, Torino, Italy, April 6–8, 1982, Proceedings. Lecture Notes in Computer Science, vol 137, pp 337–351. https://doi.org/10.1007/3-540-11494-7_22. https://doi.org/10.1007/3-540-11494-7_22
3. Clarke EM, Emerson EA, Sifakis J (2009) Model checking: algorithmic verification and debugging. Commun ACM 52(11):74–84. https://doi.org/10.1145/1592761.1592781
4. Biere A, Cimatti A, Clarke EM, Strichman O, Zhu Y (2003) Bounded model checking. Adv Comput 58:117–148
5. Sheeran M, Singh S, Stålmarck G (2000) Checking safety properties using induction and a sat-solver. In: FMCAD. Lecture notes in computer science, vol 1954, pp 108–125
6. McMillan KL (2003) Interpolation and sat-based model checking. In: CAV. Lecture notes in computer science, vol 2725, pp 1–13
7. Bradley AR (2011) SAT-based model checking without unrolling. In: VMCAI. LNCS, vol 6538, pp 70–87
8. Clarke EM, Grumberg O, Jha S, Lu Y, Veith H (2003) Counterexample-guided abstraction refinement for symbolic model checking. J ACM 50(5):752–794. https://doi.org/10.1145/876638.876643
9. Tonetta S (2009) Abstract model checking without computing the abstraction. In: FM. Lecture notes in computer science, vol 5850, pp 89–105
10. Cimatti A, Griggio A, Mover S, Tonetta S (2016) Infinite-state invariant checking with IC3 and predicate abstraction, vol 49, pp 190–218
11. Cimatti A, Griggio A, Irfan A, Roveri M, Sebastiani R (2018) Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions. ACM Trans Comput Log 19(3):19–11952. https://doi.org/10.1145/3230639
12. Daniel J, Cimatti A, Griggio A, Tonetta S, Mover S (2016) Infinite-state liveness-to-safety via implicit abstraction and well-founded relations. In: CAV (1). Lecture notes in computer science, vol 9779, pp 271–291
13. Cimatti A, Griggio A, Mover S, Tonetta S (2014) Verifying LTL properties of hybrid systems with K-liveness. In: CAV. Lecture notes in computer science, vol 8559, pp 424–440
14. Cimatti A, Griggio A, Magnago E (2021) Proving the existence of fair paths in infinite-state systems. In: VMCAI. Lecture notes in computer science, vol 12597, pp 104–126
15. Cimatti A, Griggio A, Magnago E (2021) Automatic discovery of fair paths in infinite-state transition systems. In: ATVA. Lecture notes in computer science, vol 12971, pp 32–47
16. Cavada R, Cimatti A, Dorigatti M, Griggio A, Mariotti A, Micheli A, Mover S, Roveri M, Tonetta S (2014) The nuxmv symbolic model checker. In: CAV. Lecture notes in computer science, vol 8559, pp 334–342

17. Cimatti A, Clarke EM, Giunchiglia E, Giunchiglia F, Pistore M, Roveri M, Sebastiani R, Tacchella A (2002) Nusmv 2: an opensource tool for symbolic model checking. In: CAV. Lecture notes in computer science, vol 2404, pp 359–364
18. Cimatti A, Griggio A, Schaafsma BJ, Sebastiani R (2013) The MathSAT5 SMT Solver. In: Piterman N, Smolka SA (eds) TACAS. LNCS, vol 7795, pp 93–107
19. Cimatti A, Griggio A, Tonetta S (2021) The VMT-LIB language and tools. CoRR arXiv:abs/2109.12821
20. Cimatti A, Griggio A, Mover S, Tonetta S (2015) Hycomp: an smt-based model checker for hybrid systems. In: TACAS. Lecture notes in computer science, vol 9035, pp 52–67
21. Bozzano M, Cimatti A, Gario M, Jones D, Mattarei C (2021) Model-based safety assessment of a triple modular generator with xsap. Formal Aspects Comput 33(2):251–295
22. Cimatti A, Dorigatti M, Tonetta S (2013) OCRA: a tool for checking the refinement of temporal contracts. In: ASE. IEEE, pp 702–705
23. Pakonen A (2021) Model-checking infinite-state nuclear safety i &c systems with nuxmv. In: INDIN. IEEE, pp 1–6
24. Aluf-Medina M, Korten T, Raviv A, Jr, DVN, Kugler H (2021) Formal semantics and verification of network-based biocomputation circuits. In: VMCAI. Lecture notes in computer science, vol 12597, pp 464–485
25. Amendola A, Becchi A, Cavada R, Cimatti A, Griggio A, Scaglione G, Susi A, Tacchella A, Tessi M (2020) A model-based approach to the design, verification and deployment of railway interlocking system. In: ISoLA (3). Lecture notes in computer science, vol 12478, pp 240–254
26. Limbrée C, Cappart Q, Pecheur C, Tonetta S (2016) Verification of railway interlocking: compositional approach with OCRA. In: RSSRail. Lecture notes in computer science, vol 9707, pp 134–149
27. Bozzano M, Cimatti A, Pires AF, Jones D, Kimberly G, Petri T, Robinson R, Tonetta S (2015) Formal design and safety analysis of AIR6110 wheel brake system. In: CAV (1). Lecture notes in computer science, vol 9206, pp 518–535
28. Gario M, Cimatti A, Mattarei C, Tonetta S, Rozier KY (2016) Model checking at scale: automated air traffic control design space exploration. In: CAV (2). Lecture notes in computer science, vol 9780, pp 3–22
29. Alaña E, Naranjo H, Yushtein Y, Bozzano M, Cimatti A, Gario M, de Ferluc E, Garcia G (2012) Automated generation of FDIR for the compass integrated toolset (AUTOGEF). DASIA 2012
30. Sahu S, Schorr R, Medina-Bulo I, Wagner MF (2020) Model translation from papyrus-rt into the nuxmv model checker. In: SEFM. Lecture notes in computer science, vol 12524, pp 3–20
31. Gidey HK, Collins A, Marmsoler D (2019) Modeling and verifying dynamic architectures with factum studio. In: FACS. Lecture notes in computer science, vol 12018, pp 243–251
32. Bukhari SAA, Khalid F, Hasan O, Shafique M, Henkel J (2020) Toward model checking-driven fair comparison of dynamic thermal management techniques under multithreaded workloads. IEEE Trans Comput Aided Des Integr Circuits Syst 39(8):1725–1738
33. Tseitin GS (1968) On the complexity of derivation in propositional calculus. Stud Constr Math Math Logic 2:115–125
34. Enderton HB (2001) A mathematical introduction to logic, 2nd edn. Academic Press
35. Marques-Silva J, Lynce I, Malik S (2009) Conflict-driven clause learning sat solvers. Handb Satisfiabil 185
36. Sebastiani R (2007) Lazy satisfiability modulo theories. J Satisfiabil Boolean Model Comput JSAT 3(3–4):141–224
37. Barrett CW, Sebastiani R, Seshia SA, Tinelli C (2009) Satisfiability modulo theories. In: Biere A, Heule M, van Maaren H, Walsh T (eds) Handbook of satisfiability. Frontiers in artificial intelligence and applications, vol 185. IOS Press, pp 825–885
38. Nieuwenhuis R, Oliveras A, Tinelli C (2006) Solving SAT and SAT Modulo Theories: from an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). J ACM 53(6):937–977. https://doi.org/10.1145/1217856.1217859
39. Pnueli A (1977) The temporal logic of programs. In: FOCS, pp 46–57
40. Cimatti A, Griggio A, Magnago E, Roveri M, Tonetta S (2020) Smt-based satisfiability of first-order LTL with event freezing functions and metric operators. Inf Comput 272:104502
41. Vardi MY (1995) An automata-theoretic approach to linear temporal logic. In: Banff higher order workshop, pp 238–266
42. Claessen K, Eén N, Sterin B (2013) A circuit approach to LTL model checking. In: FMCAD. IEEE, pp 53–60
43. Clarke EM, Grumberg O, Hamaguchi K (1997) Another look at LTL model checking. Formal Methods Syst Design 10(1):47–71

44. de Moura LM, Rueß H, Sorea M (2002) Lazy theorem proving for bounded model checking over infinite domains. In: CADE. Lecture notes in computer science, vol 2392, pp 438–455
45. Biere A, Artho C, Schuppan V (2002) Liveness checking as safety checking. Electron Not Theor Comput Sci 66(2):160–177
46. Claessen K, Sörensson N (2012) A liveness checking algorithm that counts. In: Cabodi G, Singh S (eds) FMCAD. IEEE, pp 52–59
47. Alur R, Dill DL (1991) The theory of timed automata. In: REX Workshop. Lecture notes in computer science, vol 600, pp 45–73
48. Kloos J, Majumdar R, Niksic F, Piskac R (2013) Incremental, inductive coverability. In: CAV. Lecture notes in computer science, vol 8044, pp 158–173
49. Kindermann R, Junttila TA, Niemelä I (2012) Smt-based induction methods for timed systems. In: FORMATS. Lecture notes in computer science, vol 7595, pp 171–187
50. Clarke EM, Grumberg O, Jha S, Lu Y, Veith H (2000) Counterexample guided abstraction refinement. In: Emerson EA, Sistla AP (eds) CAV. LNCS, vol 1855, pp 154–169
51. Clarke EM, Grumberg O, Long DE (1994) Model checking and abstraction. ACM Trans Program Lang Syst 16(5):1512–1542. https://doi.org/10.1145/186025.186051
52. Graf S, Saidi H (1997) Construction of abstract state graphs with PVS. In: Grumberg O (ed) Proc. 9th international conference on computer aided verification (CAV'97). LNCS, vol 1254, pp 72–83
53. Lahiri SK, Bryant RE, Cook B (2003) A symbolic approach to predicate abstraction. In: Jr, WAH, Somenzi F (eds) Computer aided verification, 15th international conference, CAV 2003, Boulder, CO, USA, July 8–12, 2003, Proceedings. Lecture notes in computer science, vol 2725, pp 141–153. https://doi.org/10.1007/978-3-540-45069-6_15
54. Lahiri SK, Nieuwenhuis R, Oliveras A (2006) SMT techniques for fast predicate abstraction. In: Ball T, Jones RB (eds) Computer aided verification, 18th international conference, CAV 2006, Seattle, WA, USA, August 17–20, 2006, Proceedings. Lecture Notes in Computer Science, vol 4144, pp 424–437. https://doi.org/10.1007/11817963_39
55. Lahiri SK, Ball T, Cook B (2007) Predicate abstraction via symbolic decision procedures. Log Methods Comput Sci. https://doi.org/10.2168/LMCS-3(2:1)2007
56. Schrijver A (1998) Theory of linear and integer programming. Wiley, pp 155–156
57. Loos R, Weispfenning V (1993) Applying linear quantifier elimination. Comput J 36(5):450–462
58. Monniaux D (2008) A quantifier elimination algorithm for linear real arithmetic. In: Cervesato I, Veith H, Voronkov A (eds) Logic for programming, artificial intelligence, and reasoning, 15th international conference, LPAR 2008, Doha, Qatar, November 22–27, 2008. Proceedings. Lecture Notes in Computer Science, vol 5330, pp 243–257. https://doi.org/10.1007/978-3-540-89439-1_18. https://doi.org/10.1007/978-3-540-89439-1_18
59. Monniaux D (2010) Quantifier elimination by lazy model enumeration. In: Touili T, Cook B, Jackson PB (eds) Computer aided verification, 22nd international conference, CAV 2010, Edinburgh, UK, July 15–19, 2010. Proceedings. Lecture notes in computer science, vol 6174, pp 585–599. https://doi.org/10.1007/978-3-642-14295-6_51. https://doi.org/10.1007/978-3-642-14295-6_51
60. Cavada R, Cimatti A, Franzén A, Kalyanasundaram K, Roveri M, Shyamasundar RK (2007) Computing predicate abstractions by integrating bdds and SMT solvers. In: Formal methods in computer-aided design, 7th international conference, FMCAD 2007, Austin, Texas, USA, November 11–14, 2007, Proceedings, pp 69–76. IEEE Computer Society. https://doi.org/10.1109/FAMCAD.2007.35
61. Cimatti A, Franzén A, Griggio A, Kalyanasundaram K, Roveri M (2010) Tighter integration of bdds and SMT for predicate abstraction. In: Micheli GD, Al-Hashimi BM, Müller W, Macii E (eds) Design, automation and test in Europe, DATE 2010, Dresden, Germany, March 8–12, 2010. IEEE Computer Society, pp 1707–1712. https://doi.org/10.1109/DATE.2010.5457090
62. Cimatti A, Dubrovin J, Junttila TA, Roveri M (2009) Structure-aware computation of predicate abstraction. In: Proceedings of 9th international conference on formal methods in computer-aided design, FMCAD 2009, 15–18 November 2009, Austin, Texas, USA. IEEE, pp 9–16. https://doi.org/10.1109/FMCAD.2009.5351149
63. Gupta A, Strichman O (2005) Abstraction refinement for bounded model checking. In: CAV. Lecture notes in computer science, vol 3576, pp 112–124
64. Cimatti A, Griggio A, Sebastiani R (2010) Efficient generation of craig interpolants in satisfiability modulo theories. ACM Trans Comput Log 12(1):7–1754
65. Clarke EM, Grumberg O, Peled DA (2001) Model checking. MIT Press. http://books.google.de/books?id=Nmc4wEaLXFEC
66. Henzinger TA, Jhala R, Majumdar R, Sutre G (2002) Lazy abstraction. In: POPL, pp 58–70

67. Cimatti A, Griggio A, Mover S, Tonetta S (2016) Infinite-state invariant checking with IC3 and predicate abstraction. Formal Methods Syst Des 49(3):190–218. https://doi.org/10.1007/s10703-016-0257-4
68. Henzinger TA, Jhala R, Majumdar R, McMillan KL (2004) Abstractions from proofs. In: POPL, pp 232–244
69. Ball T, Podelski A, Rajamani SK (2002) Relative completeness of abstraction refinement for software model checking. In: Katoen J, Stevens P (eds) TACS. LNCS, vol 2280, pp 158–172
70. Collins GE (1975) Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Automata theory and formal languages. Lecture notes in computer science, vol 33, pp 134–183
71. Jovanovic D, de Moura LM (2012) Solving non-linear arithmetic. In: IJCAR. Lecture notes in computer science, vol 7364, pp 339–354
72. Henzinger TA (1996) The theory of hybrid automata. In: LICS. IEEE Computer Society, pp 278–292
73. Audemard G, Cimatti A, Kornilowicz A, Sebastiani R (2002) Bounded model checking for timed systems. In: FORTE. Lecture notes in computer science, vol 2529, pp 243–259
74. Niebert P, Mahfoudh M, Asarin E, Bozga M, Maler O, Jain N (2002) Verification of timed automata via satisfiability checking. In: FTRTFT. Lecture notes in computer science, vol 2469, pp 225–244
75. Audemard G, Bozzano M, Cimatti A, Sebastiani R (2005) Verifying industrial hybrid systems with mathsat. Electron Not Theor Comput Sci 119(2):17–32
76. Cimatti A, Mover S, Tonetta S (2014) Quantifier-free encoding of invariants for hybrid systems. Formal Methods Syst Des 45(2):165–188
77. Cook B, Fuhs C, Nimkar K, O'Hearn PW (2014) Disproving termination with overapproximation. In: FMCAD. IEEE, pp 67–74
78. Cimatti A, Griggio A, Magnago E (2022) LTL falsification in infinite-state systems. Inf Comput 289:104977. https://doi.org/10.1016/j.ic.2022.104977
79. Cimatti A, Griggio A, Magnago E, Roveri M, Tonetta S (2019) Extending nuxmv with timed transition systems and timed temporal properties. In: CAV (1). Lecture notes in computer science, vol 11561, pp 376–386
80. Miller SP, Whalen MW, Cofer DD (2010) Software model checking takes off. Commun ACM 53(2):58–64. https://doi.org/10.1145/1646353.1646372
81. Ferrante O, Benvenuti L, Mangeruca L, Sofronis C, Ferrari A (2012) Parallel NuSMV: a NuSMV extension for the verification of complex embedded systems. In: Ortmeier F, Daniel P (eds) SAFECOMP Workshops. LNCS, vol 7613, pp 409–416
82. Cimatti A, Corvino R, Lazzaro A, Narasamdya I, Rizzo T, Roveri M, Sanseviero A, Tchaltsev A (2012) Formal verification and validation of ERTMS industrial railway train spacing system. In: Madhusudan P, Seshia SA (eds) CAV. LNCS, vol 7358, pp 378–393
83. Bozzano M, Cimatti A, Katoen J-P, Nguyen VY, Noll T, Roveri M, Wimmer R (2010) A model checker for AADL. In: Touili T, Cook B, Jackson P (eds) CAV. LNCS, vol 6174, pp 562–565
84. Chiappini A, Cimatti A, Macchi L, Rebollo O, Roveri M, Susi A, Tonetta S, Vittorini B (2010) Formalization and validation of a subset of the european train control system. In: Kramer J, Bishop J, Devanbu PT, Uchitel S (eds) ICSE (2). ACM, pp 109–118
85. Cimatti A, Griggio A, Micheli A, Narasamdya I, Roveri M (2011) Kratos: a software model checker for SystemC. In: Gopalakrishnan G, Qadeer S (eds) CAV. LNCS, vol 6806, pp 310–316
86. Bloem R, Cimatti A, Greimel K, Hofferek G, Könighofer R, Roveri M, Schuppan V, Seeber R (2010) RATSY: a new requirements analysis tool with synthesis. In: Touili T, Cook B, Jackson P (eds) CAV. LNCS, vol 6174, pp 425–429
87. Bittner B, Bozzano M, Cavada R, Cimatti A, Gario M, Griggio A, Mattarei C, Micheli A, Zampedri G (2016) The xsap safety analysis platform. In: TACAS. Lecture notes in computer science, vol 9636, pp 533–539
88. Cimatti A, Griggio A, Redondi G (2021) Universal invariant checking of parametric systems with quantifier-free SMT reasoning. In: CADE. Lecture notes in computer science, vol 12699, pp 131–147