Infinite-state liveness-to-safety via implicit abstraction and well-founded relations

Jakub Daniel[1,2⋆], Alessandro Cimatti[1], Alberto Griggio[1], Stefano Tonetta[1], and Sergio Mover[3]

[1] Fondazione Bruno Kessler
{cimatti,griggio,tonettas}@fbk.eu
[2] Charles University in Prague, Faculty of Mathematics and Physics, Department of Distributed and Dependable Systems
daniel@d3s.mff.cuni.cz
[3] University of Colorado Boulder
sergio.mover@colorado.edu

Abstract. We present a fully-symbolic LTL model checking approach for infinite-state transition systems. We extend *liveness-to-safety*, a prominent approach in the finite-state case, by means of *implicit abstraction*, to effectively prove the absence of abstract fair loops without explicitly constructing the abstract state space. We increase the effectiveness of the approach by integrating 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. The proposed approach outperforms other temporal checkers on a wide class of benchmarks.

1 Introduction

Model checking of liveness properties is a fundamental verification problem. In finite-state model checking, the most prominent approaches are liveness-to-safety (L2S) [6] and k-liveness [18], that reduce the problem to one or more safety checks. Their success is motivated by the possibility to leverage the progress of SAT-based invariant checking techniques, such as interpolation-based model checking [34] and IC3 [10].

The verification of liveness properties for *infinite-state* systems has been primarily tackled in the setting of analysis of imperative program [39, 23, 40, 30, 24, 21], or other specific classes [2, 32, 8, 25, 15]. However, in many practical cases the model is described as a symbolic transition system (e.g. [12, 35]), or it is compiled

⋆ Partially supported by the Grant Agency of the Czech Republic project 14-11384S.

into a symbolic transition system from a higher level language (e.g. networks of timed and hybrid automata [16, 43], architecture description language [4, 7, 9]).

In this paper we present a new approach for LTL model checking of infinite-state transition systems, which we call L2Sia-wfr. The approach relies on two ingredients. First, we extend *liveness-to-safety* by means of *implicit abstraction* [44]. Implicit abstraction is a form of predicate abstraction that does not require the explicit construction of the abstract transition system, and is able to deal with a large number of predicates. In this setting, implicit abstraction is key to checking the existence of abstract fair loops efficiently, given a set of predicates. Second, we integrate termination techniques based on *well-founded relations* [23]. Specifically, the technique tries to prove that any existing abstract fair loop is covered by a given set of well-founded relations. At the top level, the algorithm iterates trying to remove spurious abstract fair loops while maintaining a set of predicates and a set of well-founded relations. New predicates are added if they expose the spuriousness of the abstract path at hand by showing that its abstract prefixes can not be concretized. The set of well-founded relations is extended as a result of the analysis of the abstract lasso, guided by the construction of a ranking function. Within this framework, we also integrate k-liveness, that infers the validity of the property by proving that no path can fulfill the fairness condition more than a given number of times. As such, k-liveness is seen as a generic well-founded relation.

We implemented L2Sia-wfr on top of IC3ia, a model checking engine for safety properties that extends IC3 to the infinite-state case with the use of implicit abstraction at its core [14]. We exploit the fact that the L2Sia-wfr algorithm is highly incremental with respect to the refinement iterations to tighten the integration with IC3ia.

We carried out an experimental evaluation using liveness property benchmarks for transition systems and for imperative programs. To compare various temporal checkers, we translate transition systems to programs, and programs to transition systems. The results highlight a positive interaction between implicit abstraction and well-founded relations. Overall, L2Sia-wfr outperforms the competitor temporal checkers, not only on the benchmarks for transition systems, but also on the ones for imperative programs.

The paper is structured as follows. In Section 2 we discuss the related work. In Section 3 we present some background. In Section 4 we discuss the L2Sia-wfr approach, and in Section 5 we discuss the experimental results. In Section 6 we draw some conclusions, and outline directions for future work.

2 Related Work

The most prominent approaches to symbolic LTL model checking are based on SAT techniques and typically lift naturally to the infinite-state case using SMT solvers. k-liveness [18] remains a sound technique in the infinite-state case, although not complete since even if there is no fair path, the fairness can be visited an unbounded number of times. In this paper, we embed k-liveness as a

special case of well-founded relation based on counting the occurrences of fairness along a path.

Liveness-to-safety was extended to infinite-state systems in [42] for a number of classes of infinite-state systems, namely, (ω-)regular model checking, push-down systems, and timed automata. However, the approach is in general not sound for infinite-state transition systems, where a liveness property can be violated even if there is no lasso-shaped counterexample. In fact, in this paper, we are applying liveness-to-safety on the abstract state space, which is finite.

Predicate abstraction [28] is a general technique for model checking infinite-state systems. Once the abstract transition relation is computed, any algorithm for finite-state systems can be applied. However, on one side, the computation of the abstract state space typically blows up with few dozens of predicates, and on the other side, a finite number of predicates is not sufficient to prove the property (for example, when there are loops with counters that are not initialized). In this paper, we consider implicit abstraction to tackle the first problem and well-founded relations for the second.

The counterexample-based refinement we propose is very similar to the one presented in [3]: in both approaches, if an abstract counterexample contains a spurious prefix, new predicates are added to the abstraction, while new ranking functions are discovered in case the lasso is spurious. Our approach is completely different regarding the method used to prove the property: in [3] a ranking abstraction is used to add a monitor of a ranking function and a strong fairness on its decreasing/increasing and then conventional (i.e. "explicit") predicate abstraction is used to prove the modified liveness property; our approach is instead based on reachability analysis and a novel combination of liveness-to-safety, implicit abstraction, and well-founded relations, all tightly integrated within an efficient IC3-based algorithm.

Our algorithm presents some similarities also with the work of [24], where the abstraction is based on the control-flow graph and is refined by removing paths obtained from spurious counterexamples by generalizing infeasible prefixes or termination arguments on loops. Both techniques start from the observation that it is typically easier to refute spurious counterexamples that are due to an infeasible (and bounded) execution prefix than to syntesize a termination argument showing the infeasibility of an infinite path. However, the two approaches differ in the way this observation is turned into an actual procedure. In particular, the approach of [24] is specialised for imperative programs with an explicit control-flow graph, and is based on the construction and manipulation of Büchi automata. Our approach, instead, works on fully-symbolic transition systems, and is based on implicit predicate abstraction.

Another kind of abstraction, targeting liveness properties, is transition predicate abstraction [40]. It extends the classical predicate abstraction by annotating abstract states with abstract transitions. It builds on transition invariants [39] to reduce liveness to fair termination. The technique proves that abstract transitions are well-founded to determine termination. To prove the liveness property, the technique determines whether all fair states are terminating. Our approach

differs because it uses predicate abstraction (and well-known refinement techniques) and symbolic liveness model checking techniques such as liveness-to-safety and k-liveness.

Other techniques such as [23, 39, 11] focus on the specific problem of termination, which is reduced to a binary reachability using disjunctively well-founded invariants. They have also been extended to address temporal properties as in [21, 22].

There are several other approaches to the verification of special classes of infinite-state systems such as (ω-)regular model checking [8], push-down systems [25], timed and hybrid automata [15]. The current paper focuses on the verification of generic symbolic transition systems. Specialization of the proposed methods for the above classes is left to future work.

3 Background

Transition Systems. Our setting is standard first order logic. We use the standard notions of theory, satisfiability, validity, logical consequence. We denote formulas with $\phi, \varphi, \psi, I, T$, variables with x, y, and sets of variables with X, Y, $\overline{X}$, $\widehat{X}$. A literal is an atomic formula or its negation. Given a formula φ and the set of its atoms A, an implicant is a conjunction of literals over A that implies φ. In this paper, we shall deal with *linear arithmetic* formulas, that is, Boolean combinations of propositional variables and linear inequalities. A *transition system* S is a tuple $\langle X, I, T\rangle$ where X is a set of (state) variables, $I(X)$ is a formula representing the initial states, and $T(X, X')$ is a formula representing the transitions. Given a formula ϕ over variables X, we denote with ϕ' [$\phi^{\langle n\rangle}$, respectively] the formula obtained by replacing each $x \in X$ with x' [x with n primes] in ϕ. A *state* of S is an assignment to the variables X. A [finite] *path* of S is an infinite sequence $s_0, s_1, \ldots$ [resp., finite sequence $s_0, s_1, \ldots, s_k$] of states such that $s_0 \models I$ and, for all $i \geq 0$ [resp., $0 \leq i < k$], $s_i, s'_{i+1} \models T$. Given two transitions systems $S_1 = \langle X_1, I_1, T_1\rangle$ and $S_2 = \langle X_2, I_2, T_2\rangle$, we denote with $S_1 \times S_2$ the synchronous product $\langle X_1 \cup X_2, I_1 \wedge I_2, T_1 \wedge T_2\rangle$. A predicate is a formula over state variables. Given a set P of predicates and a path $\pi \stackrel{\text{def}}{=} s_0, s_1, \ldots$ of a transition system, we call the *abstraction of π wrt. P*, denoted $\left[\widehat{\pi}\right]_P$, the sequence of sets of states $\left[\widehat{s_0}\right]_P, \left[\widehat{s_1}\right]_P, \ldots$ obtained by evaluating the predicates in P in the states of π. Given a predicate ϕ, the invariant model checking problem, denoted with $S \models_{inv} \phi$, is the problem to check if, for all finite paths $s_0, s_1, \ldots, s_k$ of S, for all i, $0 \leq i \leq k$, $s_i \models \phi$.

Linear-time Temporal Logic verification using symbolic automata-based techniques. We use the standard notions of Linear-time Temporal Logic (LTL) formulas and their semantics wrt. infinite paths of a symbolic transition system, as can be found e.g. in [37]. We denote temporal operators in boldface (e.g. **F** for **F**inally, **G** for **G**lobally, **X** for ne**X**t, **U** for **U**ntil). Given a transition system $S \stackrel{\text{def}}{=} \langle X, I, T\rangle$ and an LTL formula ϕ over X, we focus on the

model checking problem of finding if, for all infinite paths π of S, ϕ is true in π. We denote this with $S \models \phi$. The automata-based approach [45] to LTL model checking consists of building a transition system $S_{\neg\phi}$ with a fairness condition $f_{\neg\phi}$, such that $S \models \phi$ iff $S \times S_{\neg\phi} \models \mathbf{FG}\neg f_{\neg\phi}$. Showing that $S \not\models \phi$ amounts to find a counterexample in the form of a fair path, i.e., a path that visits the $f_{\neg\phi}$ infinitely many times. In finite-state systems, if ϕ does not hold, there is always a counterexample in lasso-shaped form, i.e., formed by a prefix (or stem) and a loop. In the following, without loss of generality, we assume that the automata-based transformation has been applied to the LTL 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 linear arithmetic (in)equalities. When clear from the context, we also drop the subscript $\cdot_{\neg\phi}$, and simply use $\mathbf{FG}\neg f$.

Liveness to safety (L2S). The *liveness to safety reduction* (L2S) [6] is a technique for reducing an LTL model checking problem on a finite-state transition system to an invariant model checking problem. The idea is to encode as an invariant property the absence of a lasso-shaped path violating the property $\mathbf{FG}\neg f$. This is achieved by transforming the original transition system S to the transition system S_{L2S}, introducing a set $\overline{X}$ of variables containing a copy $\overline{x}$ for each state variable x of the original system, plus additional variables *seen*, *triggered* and *loop*. Let $S \stackrel{\text{def}}{=} \langle X, I, T\rangle$. L2S transforms the transition system in $S_{\text{L2S}} \stackrel{\text{def}}{=} \langle X \cup X_{\text{L2S}}, I_{\text{L2S}}, T_{\text{L2S}}\rangle$ so that $S \models \mathbf{FG}\neg f$ if and only if $S_{\text{L2S}} \models \neg bad_{\text{L2S}}$, where:

$$
\begin{aligned}
X_{\text{L2S}} &\stackrel{\text{def}}{=} \{seen, triggered, loop\} \cup \overline{X} \\
I_{\text{L2S}} &\stackrel{\text{def}}{=} I \wedge \neg seen \wedge \neg triggered \wedge \neg loop \\
T_{\text{L2S}} &\stackrel{\text{def}}{=} T \wedge \left[\textstyle\bigwedge_X \overline{x} \leftrightarrow \overline{x}'\right] \wedge \left[seen' \leftrightarrow (seen \vee \textstyle\bigwedge_X (x \leftrightarrow \overline{x}))\right] \\
&\quad \wedge \left[triggered' \leftrightarrow (triggered \vee (f \wedge seen'))\right] \\
&\quad \wedge \left[loop' \leftrightarrow (triggered' \wedge \textstyle\bigwedge_X (x' \leftrightarrow \overline{x}'))\right] \\
bad_{\text{L2S}} &\stackrel{\text{def}}{=} loop
\end{aligned}
$$

The variables $\overline{X}$ are used to non-deterministically guess a state of the system from which a reachable fair loop starts. The additional variables are used to remember that the guessed state was seen once and that the signal f was true at least once afterwards.

Termination via disjunctive well-founded transition invariants. If S is a transition system with reachable states R and transition relation T, a relation ρ over the states of S is said to be a *transition invariant* if it contains the transitive closure of T restricted to the states in R (i.e. $T^+ \cap (R \times R) \subseteq \rho$) [39]. A binary relation $\rho \subseteq Q \times Q$ is *well-founded* if every non-empty subset $U \subseteq Q$ has a minimal element wrt. ρ, i.e. there is $m \in U$ such that no $u \in U$ satisfies $\rho(u, m)$. A relation is said to be *disjunctively well-founded* if it is a finite union of well-founded relations. Termination of a program can be reduced to finding a (disjunctively)

well-founded transition invariant for it. The technique of [23] reduces the problem of finding a disjunctively well-founded transition invariant to the verification of invariant properties. It works on imperative programs with an explicit representation of the control-flow graph (although in principle it can be applied also in a fully-symbolic setting). A given set of well-founded relations W is conjectured to be a transition invariant for the program. This condition is encoded as an invariant property, and checked with an off-the-shelf invariant-checking engine. In case of failure, the encoding is refined in a counterexample-guided manner by adding new well-founded relations to W, obtained by synthesizing ranking functions for potentially non-terminating lasso-shaped paths in the control-flow graph.

IC3 with implicit abstraction. IC3 [10] is a SAT-based algorithm for the verification of invariant properties of transition systems. It incrementally builds an inductive invariant for the property by discovering relatively-inductive formulas obtained by generalization while disproving candidate counterexamples. A a novel approach to lift IC3 to the SMT case has been recently presented in [14]. The technique is able to deal with infinite-state systems by means of a tight integration with *predicate abstraction* [28]. The approach leverages *Implicit Abstraction* [44] to express abstract transitions without computing explicitly the abstract system, and is fully incremental with respect to the addition of new predicates. The main idea of IC3 with Implicit Abstraction (IC3IA) is that of replacing the *relative induction* check of IC3, which is the central operation of the algorithm, with an *abstract* version, defined using implicit abstraction. Given a transition system $S = \langle X, I, T \rangle$, a formula $\varphi(X)$ representing a set of states, and an overapproximation $F(X)$ of states of S reachable in up to k steps, a relative induction query determines whether φ is inductive relative to F, by checking if the formula $\varphi(X) \wedge F(X) \wedge T(X, X') \wedge \neg\varphi(X')$ is unsatisfiable. Given a set of state predicates $P = \{p_i(X)\}_i$, and assuming that both F and φ are Boolean combinations of predicates from P, the corresponding abstract query is the check for unsatisfiability of the following SMT formula:

$$\varphi(X) \wedge F(X) \wedge T(Y, Y') \wedge \textstyle\bigwedge_{p_i \in P}[(p_i(X) \leftrightarrow p_i(Y)) \wedge (p_i(X') \leftrightarrow p_i(Y'))] \wedge \neg\varphi(X')$$

where Y, Y' are sets of fresh variables. Using the above, IC3 can be generalized from SAT to SMT with very little effort, at the cost of introducing spurious error paths. When this happens, the abstraction can be refined in a counterexample-guided manner, using standard techniques for extracting new predicates (e.g. interpolation). The loop continues until either a real counterexample is found (and so the property does not hold), no more counterexamples are found (and so the abstraction is precise enough to conclude that the property holds), or resources (e.g. time or memory) are exhausted. We refer the reader to [14] for more details about IC3IA.

4 Liveness-to-safety for infinite-state systems

Top-level Algorithm. We reduce the problem of checking an LTL property $\mathbf{FG}\neg f$ on a transition system S to a sequence of invariant checking problems $S_0 \models_{inv} \phi_0$, $S_1 \models_{inv} \phi_1$, ... 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: $S_j, \phi_j := \textsc{encode}(S, f, P, W)$. ENCODE ensures that if $S_j \models_{inv} \phi_j$, then $S \models \mathbf{FG}\neg f$, in which case the iteration terminates. If $S_j \not\models_{inv} \phi_j$, 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: $S_{j+1}, \phi_{j+1} := \textsc{encode}(S, f, P \cup P', W \cup W')$, where $P', W' := \textsc{refine}(S_j, \pi, P, W)$. If we can neither confirm nor refute the existence of real counterexamples, we abort the execution, returning "unknown". We might also diverge and/or exhaust resources in various intermediate steps (e.g. in checking $S_j \models \phi_j$ or during refinement). In the following, we describe in detail our encoding and refinement procedures. We begin in §4.1 with a simplified version that only uses predicates from P, i.e. $W = \emptyset$. We then describe how to extend the encoding to exploit also well-founded relations in §4.2.

4.1 Liveness-to-safety with implicit abstraction

Our first contribution is an extension of the L2S reduction described above to the infinite-state case. We first note that the original L2S transformation is not sound for infinite-state systems. This is because L2S produces a transition system that reaches the error condition if and only if there exists a lasso-shaped path. While for finite-state systems it is enough to consider lasso-shaped counterexamples, this is not true in the infinite-state case. Consider the transition system $S = \langle \{x\}, x = 0, x' = x + 1 \rangle$, with x integer, and the property $\phi \stackrel{\text{def}}{=} \mathbf{FG}(x < 5)$. Clearly, ϕ does not hold in S. Suppose to apply the L2S transformation to S as described in the previous Section, obtaining the transition system S_{L2S}. Since in S there are no lasso-shaped paths, in S_{L2S} there are no paths such that the value of $\overline{x}$ (the copy of the variable x introduced in S_{L2S}) is equal to x. Thus, S_{L2S} is safe even if $S \not\models \phi$.

We overcome this problem by incorporating implicit abstraction in the L2S encoding. Intuitively, the idea is to search for a sufficiently precise predicate abstraction of the original system, if there is one, that does not admit a path visiting the fairness condition an infinite number of times. This exploits the fact that predicate abstraction preserves the validity of universal properties [19] (therefore, if a property $\mathbf{FG}\neg f$ holds in the abstract system, then it holds also in the concrete one).

In fact, we do not need to compute the full predicate abstraction of the system S. Instead, we characterize abstract paths directly in the property, by reducing the LTL model checking problem on $\mathbf{FG}\neg f$ to proving the absence

of paths with an *abstract fair loop*. Given a set of predicates $P = \{p_i(X)\}_i$, the encoding consists of storing only the truth assignments to the predicates non-deterministically, and detecting a loop if the system visits again the same abstract state, with the fairness signal f satisfied at least once in the loop. More specifically, from a system $S = \langle X, I, T\rangle$, a property $\mathbf{FG}\neg f$, and a set of predicates P, our abstract L2S transformation (αL2S) produces a system $S_{\alpha\text{L2S}} = \langle X \cup X_{\alpha\text{L2S}}, I_{\alpha\text{L2S}}, T_{\alpha\text{L2S}}\rangle$ and an invariant property $\neg bad_{\alpha\text{L2S}}$ as follows:

$$
\begin{aligned}
X_{\alpha\text{L2S}} &\stackrel{\text{def}}{=} \{seen, triggered, loop\} \cup \{c_{p_i} \mid p_i \in P\} \\
I_{\alpha\text{L2S}} &\stackrel{\text{def}}{=} I \wedge \neg seen \wedge \neg triggered \wedge \neg loop \\
T_{\alpha\text{L2S}} &\stackrel{\text{def}}{=} T \wedge \big[\textstyle\bigwedge_{p_i\in P} c_{p_i} \leftrightarrow c'_{p_i}\big] \wedge \big[seen' \leftrightarrow (seen \vee \textstyle\bigwedge_{p_i\in P}(p_i \leftrightarrow c_{p_i}))\big] \\
&\quad \wedge \big[triggered' \leftrightarrow (triggered \vee (f \wedge seen'))\big] \\
&\quad \wedge \big[loop' \leftrightarrow (triggered' \wedge \textstyle\bigwedge_{p_i\in P}(p'_i \leftrightarrow c'_{p_i}))\big] \\
bad_{\alpha\text{L2S}} &\stackrel{\text{def}}{=} loop
\end{aligned}
$$

where $X_{\alpha\text{L2S}}$ is a set of fresh Boolean variables.

Theorem 1 (αL2S soundness). *Let $S = \langle X, I, T\rangle$ be a transition system, P a set of predicates over X, and ψ an LTL property $\mathbf{FG}\neg f$. Let the system $S_{\alpha\text{L2S}}$ and invariant property $\neg bad_{\alpha\text{L2S}}$ be the results of applying αL2S to S and ψ. Then $S_{\alpha\text{L2S}} \models_{inv} \neg bad_{\alpha\text{L2S}}$ only if $S \models \psi$.*

Proof. First, we observe that all initial states I are represented in $I_{\alpha\text{L2S}}$, and that if we take any state s of S and any successor s' of s under T then there exist corresponding states $s_{\alpha\text{L2S}}$ and $s'_{\alpha\text{L2S}}$ related by $T_{\alpha\text{L2S}}$. This can be seen by noticing that the extra constraints added to T to obtain $T_{\alpha\text{L2S}}$ do not restrict the values of the original variables of the system. Therefore, we can lift the correspondence relation to paths, and conclude that every path π of S corresponds to at least one path of $S_{\alpha\text{L2S}}$.

Suppose now by contradiction that $S_{\alpha\text{L2S}} \models_{inv} \neg bad_{\alpha\text{L2S}}$, but $S \not\models \mathbf{FG}\neg f$. Then, there exists an infinite path $\pi \stackrel{\text{def}}{=} s_0, s_1, \ldots$ of S in which f holds infinitely-often. Let $\widehat{\pi} \stackrel{\text{def}}{=} \widehat{s_0}, \widehat{s_1}, \ldots$ be a path in $S_{\alpha\text{L2S}}$ corresponding to π, and let $\big[\widehat{\pi}\big]_P \stackrel{\text{def}}{=} \big[\widehat{s_0}\big]_P, \big[\widehat{s_1}\big]_P, \ldots$ be its abstraction wrt. P. Since P is finite, so is the number of different states in $\big[\widehat{\pi}\big]_P$. Let i be a position in $\big[\widehat{\pi}\big]_P$ such that all different abstract states occur at least once in $\big[\widehat{s_0}\big]_P, \ldots, \big[\widehat{s_i}\big]_P$. Let $j > i$ be a position in π such that $s_j \models f$. Since π is infinite and f holds infinitely-often, j must exist. Then, $\big[\widehat{s_j}\big]_P$ must be equal to one of the states in $\big[\widehat{s_0}\big]_P, \ldots, \big[\widehat{s_i}\big]_P$, and therefore $\widehat{s_0}, \ldots, \widehat{s_j}$ is a counterexample for $\neg bad_{\alpha\text{L2S}}$ in $S_{\alpha\text{L2S}}$, which contradicts our initial assumption. □

Counterexamples and refinement. If $\neg bad_{\alpha\text{L2S}}$ holds in $S_{\alpha\text{L2S}}$, then $S \models \mathbf{FG}\neg f$. The converse, however, is not true. A counterexample path leading to $bad_{\alpha\text{L2S}}$ in $S_{\alpha\text{L2S}}$ might correspond to a real counterexample in S, but it might

also be due to an insufficient precision of the abstraction induced by the predicates P. We deal with this case using the following counterexample-guided refinement step. A violation of the property $bad_{\alpha\text{L2S}}$ in $S_{\alpha\text{L2S}}$ implies that the counterexample path forms a lasso in the abstract state space induced by the predicates P. We first search for a concrete lasso witnessing a real violation of the LTL property, using standard bounded model checking. If this fails, we try to prove that the abstract lasso is infeasible. We check an increasing number of finite unrollings of the lasso, and, upon infeasibility, we extract new predicates from sequence interpolants, similarly to popular refinement strategies used for invariant properties (e.g. [31]). More specifically, let $\pi \stackrel{\text{def}}{=} s_0, \ldots, s_l, \ldots, s_{l+k}$ be a finite path in $S_{\alpha\text{L2S}}$ such that $s_{l+k} \models bad_{\alpha\text{L2S}}$, $s_{l-1} \models \neg seen$ and $s_l \models seen$. Let $\left[\widehat{\pi}\right]_P$ be the abstraction of π wrt. P. We search for the smallest integer $i \geq 0$ such that the formula

$$\begin{aligned} &\underbrace{I \wedge \left[\widehat{s_0}\right]_P \wedge T}_{\phi_0} \wedge \ldots \wedge \underbrace{\left[\widehat{s_{l-1}}\right]_P^{\langle l-1\rangle} \wedge T^{\langle l-1\rangle}}_{\phi_{l-1}} \wedge \textstyle\bigwedge_{j=0}^{i} \Big[\underbrace{\left[\widehat{s_l}\right]_P^{\langle l+j\cdot k\rangle} \wedge T^{\langle l+1+j\cdot k\rangle}}_{\phi_{l+1+j\cdot k}} \wedge \ldots \\ &\qquad \ldots \wedge \underbrace{\left[\widehat{s_{l+k-1}}\right]_P^{\langle l+k-1+j\cdot k\rangle} \wedge T^{\langle l+k-1+j\cdot k\rangle}}_{\phi_{l+k-1+j\cdot k}} \Big] \wedge \underbrace{\left[\widehat{s_{l+k}}\right]_P^{\langle l+k+i\cdot k\rangle}}_{\phi_n} \end{aligned} \tag{1}$$

is unsatisfiable. If i exists, we then use an interpolating SMT solver to produce a sequence interpolant $\iota_1, \ldots, \iota_{n-1}$ for $\phi_0, \ldots, \phi_n$, and extract the set P^ι of all the atomic predicates in $\iota_1, \ldots, \iota_{n-1}$, to be added to the set of predicates used at the next iteration.

4.2 Extending liveness-to-safety with well-founded relations

The abstract L2S transformation described above is inherently limited in the kind of properties it can prove. This is not only due to the potential divergence of the refinement loop, which is a possibility shared by all approaches based on predicate abstraction applied to undecidable problems, but also to the fact that a single refinement operation may not terminate. If the abstract counterexample that is being simulated can be concretized only with paths that are not in a lasso-shaped form, then (1) will always be satisfiable. However, refinement might not terminate even if the abstract path cannot be concretized. This happens in all cases in which there is no feasible concrete path that executes the abstract loop an infinite number of times, but all finite unrollings of the loop are instead concretizable. As an example, consider the system $S \stackrel{\text{def}}{=} \langle \{x_1, x_2\}, (x_1 = 0) \wedge (x_2 \geq 0), (x_2' = x_2) \wedge (x_1' = x_1 + 1) \rangle$ with x_1, x_2 integers, the property $\psi \stackrel{\text{def}}{=} \mathbf{FG}(x_1 > x_2)$, and the predicates $P \stackrel{\text{def}}{=} \{(x_1 \leq x_2), (0 \leq x_2), (x_1 = 0)\}$. The property holds, but αL2S will not be able to prove it. In fact, αL2S admits an abstract path with a self loop on the abstract state $\langle (x_1 \leq x_2), (0 \leq x_2), \neg(x_1 = 0) \rangle$. In this case, the unrolling of the abstract loop will not terminate: any finite path of S in which the abstract loop is unrolled i times is feasible, e.g. by starting from the initial state $\langle x_1 = 0, x_2 = i + 1 \rangle$.

We address the problem of abstract counterexamples whose finite prefixes are all feasible by extending our encoding to incorporate well-founded relations. The intuitive idea is to prove that such abstract counterexamples are spurious and to block them by finding suitable termination arguments in the form of (disjunctively) well-founded transition invariants. The extended αL2S reduction with well-founded relations, denoted with $\alpha\text{L2S}_\downarrow$, takes as input a transition system $S \stackrel{\text{def}}{=} \langle X, I, T\rangle$, an LTL property $\mathbf{FG}\neg f$, a set P of state predicates, and a set W of well-founded binary relations. The encoding extends αL2S producing a transition system $S_{\alpha\text{L2S}_\downarrow} \stackrel{\text{def}}{=} \langle X_{\alpha\text{L2S}_\downarrow}, I_{\alpha\text{L2S}_\downarrow}, T_{\alpha\text{L2S}_\downarrow}\rangle$ and an invariant property $\neg bad_{\alpha\text{L2S}_\downarrow}$ defined as:

$$
\begin{array}{rcl}
X_{\alpha\text{L2S}_\downarrow} & \stackrel{\text{def}}{=} & X \cup X_{\alpha\text{L2S}} \cup \{x_0, \overline{x} \mid x \in X\} \cup \{r, s, w\} \\
I_{\alpha\text{L2S}_\downarrow} & \stackrel{\text{def}}{=} & I_{\alpha\text{L2S}} \wedge \bigwedge_{x\in X}(x_0 = x) \wedge r \wedge \neg s \wedge w \\
T_{\alpha\text{L2S}_\downarrow} & \stackrel{\text{def}}{=} & T_{\alpha\text{L2S}} \wedge \bigwedge_{x\in X}(x'_0 = x_0) \wedge \big[w' \leftrightarrow (w \wedge (f \rightarrow r))\big] \wedge T_{mem} \wedge T_{check} \\
T_{mem} & \stackrel{\text{def}}{=} & \big[(s \leftrightarrow s') \wedge \bigwedge_{x\in X}(\overline{x}' = \overline{x})\big] \\
 & & \vee\big[(seen \wedge \neg s \wedge s' \wedge f) \wedge \bigwedge_{x\in X}(\overline{x}' = x)\big] \\
T_{check} & \stackrel{\text{def}}{=} & r' \leftrightarrow \big[r \wedge \big((s' \wedge f') \rightarrow \bigvee W'\big)\big] \\
bad_{\alpha\text{L2S}_\downarrow} & \stackrel{\text{def}}{=} & (loop \wedge \neg w)
\end{array}
$$

where r, s, and w are additional auxiliary Boolean variables, and for every variable $x \in X$, two fresh variables x_0 and $\overline{x}$ are introduced, representing the initial value of x and a stored value of x at some previous occurrence of f respectively.

Intuitively, we weaken the invariant $\neg bad_{\alpha\text{L2S}}$ of §4.1 by allowing abstract loops with the fairness f to occur as long as w holds. The variable w initially holds and can change its phase at most once, when the current valuation of X and the stored valuation of $\overline{X}$ do not satisfy any of the relations in W. In the subformula T_{check}, the variable r captures the truth value of this test and the variable s ensures the test is carried out only when the valuation of $\overline{X}$ captures some previous valuation of X stored non-deterministically by the subformula T_{mem}.

Theorem 2 (Soundness). *Assuming a fixed finite collection W of well-founded relations, if $S_{\alpha\text{L2S}_\downarrow} \models_{inv} \neg bad_{\alpha\text{L2S}_\downarrow}$ then $S \models \mathbf{FG}\neg f$.*

In order to prove the Theorem, we need the following lemma.

Lemma 1. *For any infinite suffix π_+ of a path π satisfying $\mathbf{GF}f$ and any finite set W of well-founded relations there is a pair of states π_{+1} and π_{+2} satisfying f such that (π_{+1}, π_{+2}) is not in any relation in W.*

Proof. Let π_f denote the transitive closure of π_+ restricted to states where f holds, i.e. an infinite graph over nodes corresponding to those states of π_+ that satisfy f and edges connecting nodes π_{f_i} and π_{f_j} when the latter is reachable from the former. Because all the states lie on a single straight path, the graph is complete. And suppose all edges (π_{f_i}, π_{f_j}) are covered by W. By Ramsey's theorem [41] there exists a complete infinite subgraph π_{W_k} of π_f whose edges are

all covered by $W_k \in W$ for some k. The subgraph π_{W_k} forms an infinite chain in the set of states in the ordering imposed by the path π_+. This is in conflict with the fact that all the relations in W are well-founded and thus do not admit infinite chains. □

We can now prove Theorem 2.

Proof (Theorem 2). First, with an argument analogous to that used for proving Theorem 1, we can conclude that every path π of S corresponds to at least one path of $S_{\alpha\text{L2S}_\downarrow}$.

Let us assume (by contradiction) $S_{\alpha\text{L2S}_\downarrow}$ is safe and $S \not\models \mathbf{FG}\neg f$, then there exists a path $\pi \stackrel{\text{def}}{=} s_0, s_1, \ldots$ of S where f appears infinitely often (the witness to violation of the property $\mathbf{FG}\neg f$). Let $\widehat{\pi} \stackrel{\text{def}}{=} \widehat{s_0}, \widehat{s_1}, \ldots$ be a path in $S_{\alpha\text{L2S}_\downarrow}$ corresponding to π and let $\left[\widehat{\pi}\right]_P$ be its abstraction wrt. P. Let $\left[\widehat{s_k}\right]_P$ be the first step where all the distinct abstract states occurring on $\left[\widehat{\pi}\right]_P$ were visited at least once. By Lemma 1 there are s_{i_2} and s_{i_3} such that $k < i_2 < i_3$ and (s_{i_2}, s_{i_3}) are not in any relation in W. But $\left[\widehat{s_{i_3}}\right]_P$ must be equal to some state $\left[\widehat{s_{i_1}}\right]_P$ for $i_1 \leq k$. Thanks to T_{mem}, $\overline{X}$ stores the valuation of X at $\widehat{s_{i_2}}$ and preserves it for the rest of the path. There exists a valuation of the variables $\{c_{p_i} \mid p_i \in P\}$ and an induced partitioning $\langle \widehat{s_0}, \ldots, \widehat{s_{i_1}}, \ldots, \widehat{s_{i_2}}, \ldots, \widehat{s_{i_3}}, \ldots \rangle$ of path $\widehat{\pi}$ such that the predicates $p_i \in P$ assume the values of the corresponding c_{p_i} at $\widehat{s_{i_1}}$ and $\widehat{s_{i_3}}$, and $f \in \widehat{s_{i_1}}, \widehat{s_{i_2}}, \widehat{s_{i_3}}$, and (s_{i_2}, s_{i_3}) is not in any relation in W. The variables *seen*, *triggered*, and s become satisfied after $\widehat{s_{i_1}}$ and the literals *loop*, $\neg w$, and $\neg r$ become satisfied at $\widehat{s_{i_3}}$. Therefore, there exists some positive integer n such that

$$I_{\alpha\text{L2S}_\downarrow} \wedge T_{\alpha\text{L2S}_\downarrow} \wedge \ldots \wedge T^{\langle n-1 \rangle}_{\alpha\text{L2S}_\downarrow} \wedge bad^{\langle n \rangle}_{\alpha\text{L2S}_\downarrow}$$

is satisfiable and thus $\widehat{\pi}$ violates the invariant property $\neg bad_{\alpha\text{L2S}_\downarrow}$ in finite number of steps, which contradicts our initial assumptions. We conclude that if $S_{\alpha\text{L2S}_\downarrow}$ satisfies the invariant property $\neg bad_{\alpha\text{L2S}_\downarrow}$ then $S \models \mathbf{FG}\neg f$. □

Counterexamples and refinement. In order to refine the extended encoding in case of spurious counterexamples, we modify the procedure described in §4.1 as follows. We first check if the set of predicates P can be refined by blocking a finite unrolling of the abstract loop (or if a real lasso-shaped counterexample can be found). We set an upper bound on the maximum number of unrollings of the abstract loop. If this bound is reached, we try to prove that no infinite unrolling of the loop exists by finding a suitable termination argument based on ranking functions. A violation of $\neg bad_{\alpha\text{L2S}_\downarrow}$ implies that the abstract counterexample path $\left[\widehat{\pi}\right]_P \stackrel{\text{def}}{=} \left[\widehat{s_0}\right]_P, \ldots, \left[\widehat{s_l}\right]_P, \ldots, \left[\widehat{s_j}\right]_P, \ldots, \left[\widehat{s_k}\right]_P, \ldots, \left[\widehat{s_{l+n}}\right]_P$ forms a lasso in the space of the predicates P, with stem $\left[\widehat{\pi_{stem}}\right]_P \stackrel{\text{def}}{=} \left[\widehat{s_0}\right]_P, \ldots, \left[\widehat{s_{l-1}}\right]_P$ and loop $\left[\widehat{\pi_{loop}}\right]_P \stackrel{\text{def}}{=} \left[\widehat{s_l}\right]_P, \ldots, \left[\widehat{s_{l+n}}\right]_P$. Within the loop there are two distinct steps $\left[\widehat{s_j}\right]_P$ and $\left[\widehat{s_k}\right]_P$ that both satisfy f but $(\left[\widehat{s_j}\right]_P, \left[\widehat{s_k}\right]_P)$ is not in any relation in W. Let φ_{stem} and φ_{loop} be defined as:

$$\varphi_{stem} \stackrel{\text{def}}{=} \bigwedge_{i=0}^{l-1} (\left[\widehat{s_i}\right]_P^{\langle i \rangle} \wedge T^{\langle i \rangle}) \wedge \left[\widehat{s_l}\right]_P^{l} \qquad \varphi_{loop} \stackrel{\text{def}}{=} \bigwedge_{i=l+1}^{l+n} (T^{\langle i-1 \rangle} \wedge \left[\widehat{s_i}\right]_P^{\langle i \rangle})$$

where T is the transition relation of the original system S. If T doesn't contain disjunctions, then several off-the-shelf techniques for ranking function synthesis (e.g. [38, 30]) can be used for constructing a termination argument for the simple lasso represented by $\varphi_{stem} \wedge \varphi_{loop}$. However, in general T does contain disjunctions. In this case, we can enumerate the simple lassos symbolically represented by $\varphi_{stem} \wedge \varphi_{loop}$ (i.e. lassos without disjunctions), and attempt to build a termination argument for each of them. We do this using the algorithm presented in [36], a technique for enumerating an overapproximation of the prime implicants of a formula by exploiting SMT solving under assumptions. Each implicant of $\varphi_{stem} \wedge \varphi_{loop}$ corresponds to a simple lasso, for which we try to build a termination argument with the technique of [30]. In case of success, we use the ranking function $r(X)$ and its lower bound b produced by [30] to generate a well-founded relation. If this relation was not in the set W already, we add it to W, add its atomic predicates to P, stop the enumeration of simple lassos, and refine the encoding, proceeding to checking the new invariant property on the refined system.[4] Otherwise, we continue with the enumeration, until we either eventually find some new information that allows us to refine our encoding, or we fail to build a termination argument for all the simple lassos represented by the abstract counterexample. In the latter case, the algorithm is aborted, and "unknown" is returned.

***k*-liveness as a well-founded relation.** Generally, infinite-state systems admit counterexamples that are not lasso-shaped. But we analyze infeasibility of a strict subset of all possible infinite unrollings of the abstract lasso. Consequently, the refinement may fail to produce new abstraction predicates or well-founded relations as well as fail to find a concrete counterexample. However, we may use k-liveness to recover from such situations in hope to make further progress. By extending the transition system with an auxiliary integer variable k representing a counter of occurrences of f, and the necessary updates of k, the refinement may always discover a new well-founded relation $\rho(\overline{X}, X) \stackrel{\text{def}}{=} \overline{k} < k \leq n$ where $\overline{k} \in \overline{X}$ is introduced by the reduction $\alpha\text{L2S}_{\downarrow}$ and n is an arbitrary positive integer. We usually let n be the number of occurrences of f in the current counterexample. By adding these free relations we allow the procedure to progress by blocking all short spurious counterexamples as w holds for the first n steps of the re-encoded system. Notice that this is sound.

4.3 Implementation within IC3ia

In principle, the technique described in the previous Sections can be implemented on top of any off-the-shelf invariant verification algorithm. In practice, however, we exploit the features of the IC3ia algorithm of [14] to obtain an efficient

[4] An alternative heuristic could be to not stop the enumeration, and instead generate well-founded relations covering all simple lassos represented by the abstract counterexample. We use a conservative/lazy heuristic, that tries to avoid the potentially-expensive exhaustive enumeration of implicants as much as possible.

implementation, in which our liveness-to-safety encoding is tightly integrated in the incremental IC3-based invariant checking procedure and its interpolation-based abstraction refinement.

First, we observe that the interpolation-based refinement of the αL2S encoding (see §4.1) is very similar to the refinement procedure already implemented in IC3IA, making it possible to reuse most of the code. More importantly, our technique can be integrated in IC3IA in a highly incremental manner. In particular, there is no need of restarting from scratch the IC3 search at every refinement iteration; rather, all the relatively-inductive formulas discovered by IC3IA in the process of constructing an inductive invariant can be retained across refinements. This is possible because:(a) our encoding only monotonically adds constraints to the original transition system, and (b) the new safety property obtained after the $i+1$-th refinement step is weaker than the previous ones; in particular, if IC3 has concluded that no state can violate the property corresponding to the i-th refinement in k steps or less, then this is true also for the $i+1$-th property. Therefore, all the invariants on which IC3 relies for its correctness are preserved by our refinement procedure, thus allowing it to continue the search without resetting its internal state.

5 Experimental Evaluation

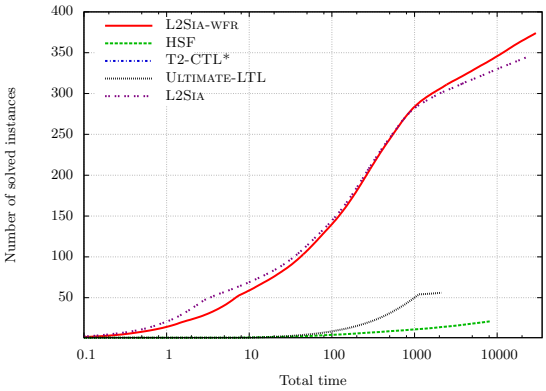
We have implemented L2SIA-WFR as an extension of the IC3IA algorithm described in [14]. The MATHSAT SMT solver [17] is used for solving SMT queries, computing interpolants, and synthesizing ranking functions (using the technique of [30]). The LTL extension to IC3IA consists of about 1500 lines of C++ code. In the evaluation, in addition to L2SIA-WFR, we consider L2SIA, i.e. the variant where well-founded relations are disabled. The source code of the implementation is available at `https://es-static.fbk.eu/people/griggio/ic3ia/`.

Tools used. We compare our implementation with the following state-of-the-art tools for temporal property verification of infinite-state systems:

HSF [29], a solver for Horn-like clauses that also supports proving well-foundedness of a given relation, using transition invariants. In order to check an LTL property φ, we apply the technique described in [39]: we encode the input transition system and a (symbolic representation of a) Büchi automaton for $\neg\varphi$ as Horn-like clauses, and ask HSF to find a (disjunctively) well-founded transition invariant showing that the accepting states of the automaton cannot be visited infinitely often. We use the LTL2BA tool of [26] to generate the Büchi automaton.

T2-CTL* [21], an extension to the T2 termination prover for imperative programs supporting CTL* temporal properties. LTL properties can be verified by simply checking the equivalent CTL* specification.[5] T2-CTL* works by recursively computing preconditions of subformulas of the input property (starting

[5] T2 supports also verification of CTL properties under fairness constraints [20], which could in principle be used for verifying LTL properties. Here we use the CTL* mode as suggested by the tool authors.



Tool	# Solved	Safe	Unsafe	$\Delta_{\text{L2Sia-wfr}}$	Gained	Lost	Cumulative time (sec)
L2Sia-wfr	374	341	33	–	–	–	29438
L2Sia	344	320	24	-30	1	31	22144
Ultimate-LTL	56	56	0	-318	0	318	2113
HSF	21	21	0	-353	0	353	8116
T2-CTL*	0	0	0	-374	0	374	0

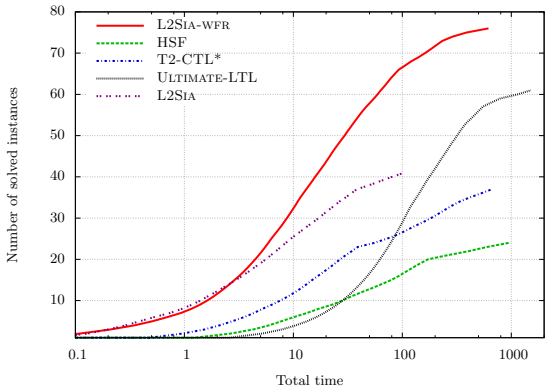
Fig. 1. Experimental results on symbolic transition systems.

from the leaves), and checking whether the precondition of the topmost formula is satisfied by the initial states of the program. Path subformulas are approximated in ACTL, the program is partially determinized using prophecy variables to reduce the imprecision of the approximation. Formula preconditions are then computed via CTL model checking. We remark that T2-CTL* is more general than L2Sia-wfr, being able to handle arbitrary CTL* properties. This fact should be considered when interpreting the experimental results presented.

Ultimate-LTL [24], a tool for the verification of LTL properties of sequential C programs. Ultimate-LTL works by enumerating the fair paths of the input program (i.e. paths visiting the accepting condition of a Büchi automaton for $\neg\varphi$ infinitely often) and trying to prove each of them unfeasible (either by refuting a finite prefix of the path, or by finding a suitable termination argument for it). If a fair path is determined to be feasible, the property is shown not to hold. Otherwise, if the fair path is successfully refuted, it is generalized and then subtracted from the input program: if the result is empty, then the property is shown to hold. Otherwise, an unknown result is reported.

Benchmark sets. Our benchmark set consists of 835 LTL verification problems, grouped in three different subsets:

Symbolic transition systems. The first set consists of a collection of symbolic transition systems: BIP models from [7], and systems derived from standard examples in the real-time domain (e.g. [5, 1, 27]) in which the variables have been converted to integers, in order to be able to use all the tools mentioned above.



Tool	# Solved	Safe	Unsafe	$\Delta_{\text{L2Sia-wfr}}$	Gained	Lost	Cumulative time (sec)
L2Sia-wfr	76	54	22	–	–	–	618
Ultimate-LTL	61	44	17	-15	2	17	1512
L2Sia	41	26	15	-35	0	35	104
T2-CTL*	36	36	0	-40	2	42	663
HSF	24	24	0	-52	0	52	951

Fig. 2. Experimental results on imperative programs.

The LTL properties have been manually generated, in order to capture standard liveness requirements on the considered domains. The set consists of 556 instances, 66 of which are unsafe. For tools working with imperative programs (T2-CTL*, Ultimate-LTL), we encode a system $S = \langle X, I, T\rangle$ as shown in the box on the right, where nondet is a function that returns a non-deterministic value. We then translate an LTL property $\mathbf{FG}\neg f$ into $\mathbf{FG}(ok \rightarrow \neg f)$. [6]

```
ok := false
∀x_i∈X x_i := nondet()
ok := true
if ¬I(X)
    ok := false
while ok
    ok := false
    ∀x_i∈X oldx_i := x_i
    ∀x_i∈X x_i := nondet()
    ok := true
    if ¬T(oldX, X)
        ok := false
```

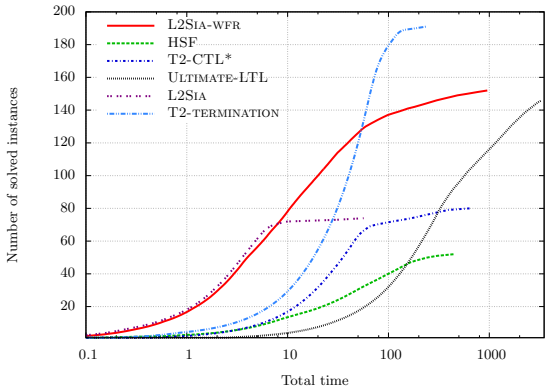
Imperative-style programs. This set consists of 86 imperative-style programs collected from three different sources:

i) 17 simple hand-crafted imperative programs with LTL properties that were specifically written to make approaches based on αL2S fail. The instances are all safe.

ii) the 41 C programs belonging to the "coolant" and "decision-predicates" groups of benchmark instances used in [24]. The programs have been translated to transition systems using the C front-end of the Kratos [13] software model checker.[7]

[6] The encoding shown is a slightly simplified one. In practice, we have experimented with several variations, and picked for each tool the encoding giving the best results.

[7] The benchmark set from [24] contains also a third group of instances ("rers2012"), which could however not be handled by the C front-end of Kratos.



Tool	# Solved	Safe	Unsafe	$\Delta_{\text{L2Sia-wfr}}$	Gained	Lost	Cumulative time (sec)
L2Sia-wfr	152	82	70	–	–	–	956
Ultimate-LTL	146	85	61	-6	15	21	3311
T2-CTL*	80	80	0	-72	6	78	638
L2Sia	74	18	56	-78	1	79	57
HSF	52	52	0	-100	1	101	442
T2-termination	*191*	*89*	*102*	*+39*	*40*	*1*	*235*

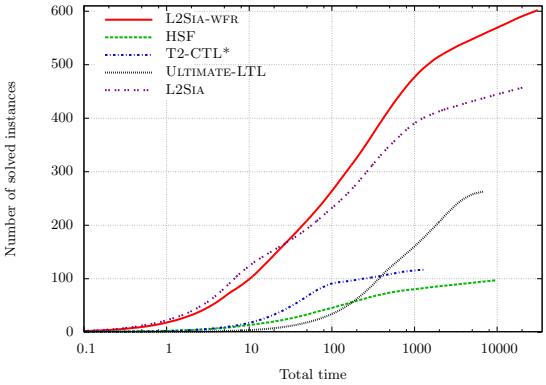
Fig. 3. Experimental results on T2 termination benchmarks.

iii) the 28 instances of the "cav13-ctl-examples" in the source distribution of T2, in which the LTL properties have been obtained by simply removing the path quantifiers from the corresponding CTL properties in the original files.

T2 termination benchmarks. The last set we considered consists of the 193 instances of the "testsuite" group in the source distribution of T2. These are termination problems that have been encoded into LTL by checking that a distinguished "sink" location is eventually reached. For this group of benchmarks, in addition to the tools described above, we compare also with the specialised procedure for termination checking in T2 [11] (called T2-termination in the following).

Results. We ran our experimental evaluation on a cluster of machines with 2.67GHz Xeon X5650 CPUs and 96Gb of RAM, running Scientific Linux 6.7. We used a timeout of 1200 seconds and a memory limit of 6Gb.

As a preliminary remark, we analyze the number of predicates used. For the verification of the benchmarks, L2Sia-wfr discovered up to 85 predicates for the αL2S$_\downarrow$ encoding and up to 278 predicates for checking the sequence of invariant properties with IC3ia, with an average of 20 predicates for αL2S$_\downarrow$ and 45 predicates for IC3ia (with median values of 14 and 34 respectively). These numbers do not include Boolean state variables in the system, that are always tracked precisely. We conclude that Implicit Abstraction is a key enabler for the proposed approach: an eager computation of the abstract transition system,



Tool	# Solved	Safe	Unsafe	$\Delta_{\text{L2Sia-wfr}}$	Gained	Lost	Cumulative time (sec)
L2Sia-wfr	602	477	125	–	–	–	31012
L2Sia	459	364	95	-143	2	145	22304
Ultimate-LTL	263	185	78	-339	17	356	6936
T2-CTL*	116	116	0	-486	8	494	1300
HSF	97	97	0	-505	1	506	9509

Fig. 4. Experimental results on all benchmarks.

based for example on AllSMT [33], is typically unable to deal with such high numbers of predicates.

The results of the evaluation are summarized in Figures 1–4. The plots show, for each tool, the number of solved instances (y-axis) in the given total execution time (x-axis), not including timeouts/unknowns. More information is provided in the tables under the plots, where for each tool we show the number of solved instances (distinguishing also between safe and unsafe ones), the difference in number of solved instances wrt. L2Sia-wfr, the number of instances gained (i.e. solved by the given tool but not by L2Sia-wfr) and lost, and the total execution time taken on solved instances. From the results, we can make the following observations:

i) L2Sia-wfr significantly outperforms all the other tools on symbolic transition systems (Fig. 1). Most of the instances are solved without the need of any ranking function, relying exclusively on abstract L2S. However, as can be seen from the comparison with L2Sia, the integration of ranking functions gives a non-negligible benefit, allowing to solve 31 instances that were out of reach before, and only losing one. It is also interesting to observe that this is beneficial not only for proving properties, but also for finding counterexamples. We attribute the performance of L2Sia-wfr to its tight integration with the engine based on IC3 with Implicit Abstraction [14], that can handle very efficiently the symbolic encodings of these benchmarks. In contrast, and as expected, tools that are optimized to exploit control-flow graphs of programs (T2-CTL*, Ultimate-LTL) perform very poorly when such information is not available.

ii) Interestingly, as shown in Fig. 2 L2Sia-wfr is the best performing tool also on the benchmarks of the imperative programs group, for which a control-flow graph is available. The gap with the other tools in this case is much smaller, and there are a number of instances which L2Sia-wfr cannot solve but some of the other tools can. However, L2Sia-wfr is still the most effective tool overall, both for safe and for unsafe instances.[8] For this set of benchmarks, the integration of ranking functions is crucial for performance.

iii) The results on termination benchmarks (Fig. 3) show that there is still a significant gap between tools supporting arbitrary LTL properties and specialised procedures for termination such as T2-termination, for which almost all these instances are very easy. Also in this case, however, L2Sia-wfr is very competitive with other tools of similar expressiveness (HSF and Ultimate-LTL).

iv) Overall (Fig. 4), L2Sia-wfr performs very well across all the categories of benchmarks we have considered, comparing very favorably with the state of the art. We think that this demonstrates the generality and potential of our approach.

6 Conclusions

In this paper we presented a novel algorithm, called L2Sia-wfr, to check liveness properties on infinite state transition systems. The algorithm combines liveness-to-safety with implicit abstraction and well-founded relations. The implementation demonstrates substantial advantages in performance over other temporal checkers for infinite-state systems.

In the future, we will explore techniques for non-termination to find counterexamples that are not lasso-shaped, thus extending the effectiveness of the algorithm in the case of property violation. Furthermore, we will investigate domain-specific techniques for the analysis of real-time/hybrid systems, such as the integration with k-zeno [15], and the extension of the class of well-founded relations over the reals. Finally, we will evaluate the application of L2Sia-wfr to temporal satisfiability of first-order temporal logic.

References

1. Alur, R., Dang, T., Ivancic, F.: Counterexample-guided predicate abstraction of hybrid systems. Theor. Comput. Sci. 354(2), 250–271 (2006)
2. Alur, R., Dill, D.L.: A Theory of Timed Automata. Theor. Comput. Sci. 126(2), 183–235 (1994)

[8] To the best of our understanding, HSF can only prove that an LTL property holds, but it is not able to find counterexamples. In principle, T2-CTL* instead should also be able to find counterexamples; however, even after asking its authors, we haven't been able to find a reliable way to distinguish an "unsafe" answer from an "unknown" one.

3. Balaban, I., Pnueli, A., Zuck, L.D.: Ranking Abstraction as Companion to Predicate Abstraction. In: FORTE. pp. 1–12 (2005)
4. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in BIP. In: SEFM. pp. 3–12. IEEE Computer Society (2006)
5. Bengtsson, J., Larsen, K.G., Larsson, F., Pettersson, P., Yi, W.: Uppaal - a tool suite for automatic verification of real-time systems. In: Hybrid Systems. pp. 232–243 (1995)
6. Biere, A., Artho, C., Schuppan, V.: Liveness Checking as Safety Checking. Electr. Notes Theor. Comput. Sci. 66(2), 160–177 (2002)
7. Bliudze, S., Cimatti, A., Jaber, M., Mover, S., Roveri, M., Saab, W., Wang, Q.: Formal Verification of Infinite-State BIP Models. In: ATVA. LNCS, vol. 9364, pp. 326–343. Springer (2015)
8. Bouajjani, A., Legay, A., Wolper, P.: Handling Liveness Properties in (ω-)Regular Model Checking. Electr. Notes Theor. Comput. Sci. 138(3), 101–115 (2005)
9. Bozzano, M., Cimatti, A., Lisagor, O., Mattarei, C., Mover, S., Roveri, M., Tonetta, S.: Safety assessment of altarica models via symbolic model checking. Sci. Comput. Program. 98, 464–483 (2015)
10. Bradley, A.R.: SAT-Based Model Checking without Unrolling. In: VMCAI. LNCS, vol. 6538, pp. 70–87. Springer (2011)
11. Brockschmidt, M., Cook, B., Fuhs, C.: Better termination proving through cooperation. In: CAV. LNCS, vol. 8044, pp. 413–429. Springer (2013)
12. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuXmv Symbolic Model Checker. In: CAV. LNCS, vol. 8559, pp. 334–342. Springer (2014)
13. Cimatti, A., Griggio, A., Micheli, A., Narasamdya, I., Roveri, M.: Kratos - A Software Model Checker for SystemC. In: CAV. LNCS, vol. 6806, pp. 310–316. Springer (2011)
14. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: IC3 Modulo Theories via Implicit Predicate Abstraction. In: TACAS. LNCS, vol. 8413, pp. 46–61. Springer (2014)
15. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: Verifying LTL Properties of Hybrid Systems with K-Liveness. In: CAV. pp. 424–440 (2014)
16. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: Hycomp: An smt-based model checker for hybrid systems. In: TACAS. LNCS, vol. 9035, pp. 52–67. Springer (2015)
17. Cimatti, A., Griggio, A., Schaafsma, B., Sebastiani, R.: The MathSAT5 SMT Solver. In: TACAS. LNCS, vol. 7795. Springer (2013)
18. Claessen, K., Sörensson, N.: A Liveness Checking Algorithm that Counts. In: FMCAD. pp. 52–59. IEEE (2012)
19. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided Abstraction Refinement for Symbolic Model Checking. J. ACM 50(5), 752–794 (2003)
20. Cook, B., Khlaaf, H., Piterman, N.: Fairness for infinite-state systems. In: TACAS. LNCS, vol. 9035, pp. 384–398. Springer (2015)
21. Cook, B., Khlaaf, H., Piterman, N.: On Automation of CTL* Verification for Infinite-State Systems. In: CAV (1). LNCS, vol. 9206, pp. 13–29. Springer (2015)
22. Cook, B., Koskinen, E., Vardi, M.Y.: Temporal Property Verification as a Program Analysis Task. In: CAV. pp. 333–348 (2011)
23. Cook, B., Podelski, A., Rybalchenko, A.: Termination proofs for systems code. In: PLDI. pp. 415–426 (2006)

24. Dietsch, D., Heizmann, M., Langenfeld, V., Podelski, A.: Fairness Modulo Theory: A New Approach to LTL Software Model Checking. In: CAV (1). LNCS, vol. 9206, pp. 49–66. Springer (2015)
25. Esparza, J., Hansel, D., Rossmanith, P., Schwoon, S.: Efficient Algorithms for Model Checking Pushdown Systems. In: CAV. pp. 232–247 (2000)
26. Gastin, P., Oddoux, D.: Fast LTL to Büchi Automata Translation. In: CAV. LNCS, vol. 2102, pp. 53–65. Springer (2001)
27. Ghilardi, S., Ranise, S.: MCMT: A model checker modulo theories. In: IJCAR. LNCS, vol. 6173, pp. 22–29. Springer (2010)
28. Graf, S., Saïdi, H.: Construction of Abstract State Graphs with PVS. In: CAV. pp. 72–83 (1997)
29. Grebenshchikov, S., Lopes, N.P., Popeea, C., Rybalchenko, A.: Synthesizing software verifiers from proof rules. In: PLDI. pp. 405–416. ACM (2012)
30. Heizmann, M., Hoenicke, J., Leike, J., Podelski, A.: Linear Ranking for Linear Lasso Programs. In: ATVA. LNCS, vol. 8172, pp. 365–380. Springer (2013)
31. Henzinger, T., Jhala, R., Majumdar, R., McMillan, K.: Abstractions from Proofs. In: POPL. pp. 232–244 (2004)
32. Henzinger, T., Kopke, P., Puri, A., Varaiya, P.: What's decidable about hybrid automata? In: STOC. pp. 373–382 (1995)
33. Lahiri, S.K., Nieuwenhuis, R., Oliveras, A.: SMT techniques for fast predicate abstraction. In: CAV. LNCS, vol. 4144, pp. 424–437. Springer (2006)
34. McMillan, K.L.: Interpolation and sat-based model checking. In: CAV. LNCS, vol. 2725, pp. 1–13. Springer (2003)
35. de Moura, L.M., Owre, S., Rueß, H., Rushby, J.M., Shankar, N., Sorea, M., Tiwari, A.: SAL 2. In: CAV. LNCS, vol. 3114, pp. 496–500. Springer (2004)
36. Niemetz, A., Preiner, M., Biere, A.: Turbo-charging lemmas on demand with don't care reasoning. In: FMCAD. pp. 179–186. IEEE (2014)
37. Pnueli, A.: The Temporal Logic of Programs. In: FOCS. pp. 46–57 (1977)
38. Podelski, A., Rybalchenko, A.: A complete method for the synthesis of linear ranking functions. In: VMCAI. LNCS, vol. 2937, pp. 239–251. Springer (2004)
39. Podelski, A., Rybalchenko, A.: Transition invariants. In: LICS. pp. 32–41. IEEE Computer Society (2004)
40. Podelski, A., Rybalchenko, A.: Transition predicate abstraction and fair termination. ACM Trans. Program. Lang. Syst. 29(3) (2007)
41. Ramsey, F.P.: On a problem in formal logic. Proc. London Math. Soc. (3) 30, 264–286 (1930)
42. Schuppan, V., Biere, A.: Liveness checking as safety checking for infinite state spaces. Electr. Notes Theor. Comput. Sci. 149(1), 79–96 (2006)
43. Tiwari, A.: Hybridsal relational abstracter. In: CAV. LNCS, vol. 7358, pp. 725–731. Springer (2012)
44. Tonetta, S.: Abstract Model Checking without Computing the Abstraction. In: FM. pp. 89–105 (2009)
45. Vardi, M.: An Automata-Theoretic Approach to Linear Temporal Logic. In: Banff Higher Order Workshop. pp. 238–266 (1995)