Automatic discovery of fair paths in infinite-state transition systems

Alessandro Cimatti[1], Alberto Griggio[1], and Enrico Magnago[1,2]

[1] Fondazione Bruno Kessler
[2] University of Trento

Abstract. Proving existential properties of infinite-state systems (e.g. software non-termination, model checking of hybrid automata) comes with a key challenge: differently from the finite-state case, witnesses may not be in form of lasso-shaped fair paths. In this paper we propose an approach to automatically prove existential properties for infinite state transition systems, presenting witnesses in an indirect way. The approach is based on the notion of *well-founded funnel*, where a ranking function guarantees that the states in the source set are guaranteed to inevitably reach the destination set. We show that, under suitable conditions, a sequence of funnels ensures the existence of a fair path. We propose an algorithm that, working in an abstract space induced by a set of predicates, identifies candidate funnels, proves their well-foundedness, and searches for a sequencing order.
An experimental evaluation shows that the approach is effective in proving existential properties on a wide range of examples taken from both software and LTL model checking, and outperforms various competitor tools.

Keywords: LTL model checking · LTL falsification · infinite-state systems · SMT.

1 Introduction

Temporal logic model checking for infinite-state transition systems is a very important direction in verification. Most of the works have been devoted to proving universal properties, i.e. properties holding on all the traces.The dual problem of proving existential properties, used for example for software non-termination and model checking of hybrid automata, comes with a fundamental difficulty: differently from the finite-state case, witnesses may not be in form of lasso-shaped fair paths.

In this paper we propose an approach to automatically prove existential properties for infinite state transition systems, presenting witnesses in an indirect way. Our approach is based on the notion of *well-founded funnel.* A (well-founded) funnel fnl comprises two sets of (source and target) states S and D, an underapproximation of the transition relation, and a ranking function proving that all paths of fnl from S will eventually reach D. A sequence of funnels $fnl_0, \ldots, fnl_{n-1}$ ensures the existence of a fair path if certain conditions are met. These include that D_{n-1} must be contained in the fairness condition, the destination D_i of fnl_i must be contained in the source S_{i+1} of the next funnel for all $i > 1$, and D_{n-1} must be contained in S_0.

We propose an algorithm that identifies candidate funnels, proves their well-foundedness, and searches for the right sequencing order so that the existence of a corresponding fair path is ensured. The algorithm works in an abstraction of the infinite-state transition system induced by a set of suitable predicates. Specifically, it uses a liveness-to-safety construction to generate lasso-shaped paths in the abstract space. At its core, the proof of well-foundedness of each funnel is carried out by synthesizing a suitable ranking function.

A key difference with respect to predicate abstraction is that here abstract traces are not required to have the same number of transitions of their concretizations – in fact, each abstract state is implicitly associated with an arbitrarily high (but finite) number of self-transitions in its concretization.

We implemented the approach in a prototype called F3, built on top of the SMT solvers MathSAT and Z3. We carried out an extensive experimental evaluation, on a wide range of examples taken from both software and LTL model checking, comparing F3 with several competitor systems. The results shows that the proposed approach has two key advantages: first, it is very general, in that none of the competitor tools is able to cover all the benchmarks; second, it is very effective in proving a large number of existential properties.

The paper is structured as follows. In Section 2 we present some preliminaries. Then, in Section 3 we define funnels and prove their properties. In Section 4 we present the algorithm, and in Section 5 we discuss the related work. In Section 6 we briefly describe some implementation details and then discuss our experimental results. In Section 7 we draw some conclusions and outline the directions for future work. The proofs of all the theorems contained in the paper are reported in the Appendix B.

2 Background

We work in the setting of SMT, with the theory of quantified real arithmetic. We assume the standard notions of interpretation, model, satisfiability, validity and logical consequence. A symbolic fair transition system M is a tuple $\langle V, I, T, F \rangle$, where V is the set of state variables; I and F are formulae over V, representing respectively the initial and fair states; T is a formula over V and V' representing the transitions, where $V' \doteq \{v' | v \in V\}$ and the primed version of a variable refers to the next state. We write $I(V)$, $F(V)$ and $T(V, V')$ to explicitly state that they are formulae over the symbols in V (I and F) and $V \cup V'$ (T) respectively.

We denote with $\boldsymbol{v}$ a total assignment over V, i.e. a state. A fair path of M is an infinite sequence of states, $\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots$, such that $\boldsymbol{v}_0 \models I$, $\boldsymbol{v}_i \boldsymbol{v}'_{i+1} \models T$ for all i, and for each i there exists $j > i$ such that $\boldsymbol{v}_j \models F$. Given a formula $\phi(V)$ we

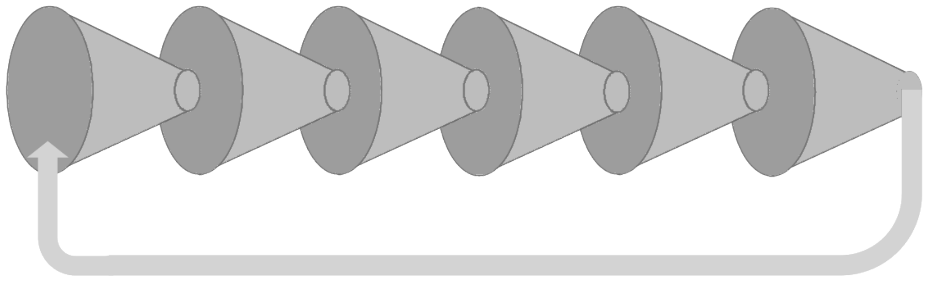

Fig. 1. Funnels combined into chain forming a funnel-loop.

also write $\phi(\boldsymbol{v})$ for the evaluation of ϕ obtained by replacing every symbol in V with its corresponding assignment in $\boldsymbol{v}$. We also assume the standard notions of trace, reachability, and temporal logic model checking, using the usual definitions of $\mathbf{U}, \mathbf{G}, \mathbf{F}$ for the "until", "always" and "eventually" temporal operators (LTL [35]).

We overload the $\models$ symbol: when ϕ and ψ are SMT formulae, then $\phi \models \psi$ stands for entailment in SMT; when M is a fair transition system and ψ is a linear temporal property, then $M \models \psi$ is to be interpreted with the LTL semantics. If ψ is a quantifier-free SMT formula and ϕ is a conjunction of (a subset of) the atoms of ψ, then ϕ is an implicant of ψ iff $\phi \models \psi$.

Given a fair transition system M, we are interested in the problem of determining whether M admits at least one fair path. Notice that the existential LTL model checking problem, i.e. the problem of determining whether a system $M \doteq \langle V, I, T, \top \rangle$ admits at least a path that satisfies a given LTL formula φ, can be reduced to checking for the existence of a fair path in the fair transition system $M \times M_\varphi \doteq \langle V \cup V_\varphi, I \wedge I_\varphi, T \wedge T_\varphi, F_\varphi \rangle$, where $M_\varphi \doteq \langle V_\varphi, I_\varphi, T_\varphi, F_\varphi \rangle$ is a symbolic encoding of an automaton accepting the language of φ [36], which can be obtained e.g. with the procedure of [10].

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 $\rho(u, m)$. Given a (transition) relation T over symbols $V \cup V'$, a ranking function $\textsc{Rf}(V)$ is a function from the assignments to the symbols V to some set Q, such that the relation $\{\langle \textsc{Rf}(\boldsymbol{v_0}), \textsc{Rf}(\boldsymbol{v_1}) \rangle \mid \boldsymbol{v_0}, \boldsymbol{v_1'} \models T\}$ is well-founded.

3 Funnels and funnel loops

We identify fair paths by means of a composition of elements called *funnels* that, like actual funnels, take items from a source and constrain them to follow a path leading to a destination. Each funnel characterizes a set of finite paths, each starting from the source region, remaining in it for a bounded number of steps, and eventually ending in the destination region. Funnels are concatenated in chains such that the destination region of a funnel is contained in the source region of the following one. Funnel-loops are chains of funnels in which the destination region of the last funnel is included in the source region of first one.

We show that under certain conditions the existence of one such funnel-loop implies the existence of a fair path for a fair transition system M.

Given a set of symbols V, a funnel is a 4-tuple $\langle S(V), T(V, V'), D(V), \textsc{Rf}(V)\rangle$. S and D are formulae representing respectively the source and destination regions, T is the transition relation and $\textsc{Rf}$ is a ranking function for S with respect to the transition relation T. Intuitively, this structure represents a terminating loop over S where D are the end states of the loop. Depending on the shape of the ranking function, the loop might correspond to a simple loop or to more complex termination arguments such as nested loops. Each path through the funnel starts from a state in S, it remains in S by following transition T while the ranking function $\textsc{Rf}$ remains greater than the minimal element $\mathbf{0}$ and finally reaches D when the ranking function becomes $\mathbf{0}$. If we consider a trivial ranking function that is always equal to the minimal element $\mathbf{0}$ the 4-tuple simply asserts that every state in S is mapped into D by a single transition T.

Definition 1 (Funnel). *Given a set of symbols V, a funnel is defined as the 4-tuple*

$$Funnel \doteq \langle S(V), T(V, V'), D(V), \textsc{Rf}(V)\rangle$$

where: S and D are SMT formulae that represent abstract states; T is a boolean formula with symbols in $V \cup V'$ over some combination of SMT-theories representing a transition relation; $\textsc{Rf}$ *is a function from the assignments to the symbols in V to some well-founded set with minimal element* $\mathbf{0}$. *Every funnel fnl satisfies the following hypotheses.*

F.1 *The transition relation is total relative to the source region.*

$$\forall V \exists V' : S(V) \to T(V, V')$$

F.2 *Every funnel keeps iterating on the source region as long as its ranking function is greater than the minimal element.*

$$\forall V, V' : (S(V) \land \textsc{Rf}(V) > \mathbf{0} \land T(V, V')) \to S(V')$$

F.3 *Every step from the source region decreases the ranking function.*

$$\forall V, V' : (S(V) \land \textsc{Rf}(V) > \mathbf{0} \land T(V, V')) \to \textsc{Rf}(V) > \textsc{Rf}(V')$$

F.4 *Once the ranking function is equal to* $\mathbf{0}$ *the funnel reaches its destination region.*

$$\forall V, V' : (S(V) \land \textsc{Rf}(V) = \mathbf{0} \land T(V, V')) \to D(V')$$

Given a funnel fnl_i we write S_i, T_i, D_i and $\textsc{Rf}_i$ to refer to its components. We define the transition system corresponding to a funnel $fnl \doteq \langle S, T, D, \textsc{Rf}\rangle$ over symbols V as $M_{fnl} \doteq \langle V, S, T, \top\rangle$. We refer to the paths through a funnel fnl with $\mathcal{L}(fnl)$ meaning the paths in the language of the corresponding transition system that end in D and write $fnl \models \phi$ meaning that ϕ holds in every path in $\mathcal{L}(fnl)$. From the definition it easily follows that every funnel fnl satisfies the following:

$$fnl \models S \ \mathbf{U} \ D$$

We define a funnel-loop as a chain of funnels $[fnl_i]_{i=0}^{n-1}$ such that the destination region of each funnel is included in the source region of the following one and the destination region of the last funnel is included in the source region of the first one.

Definition 2 (Funnel-loop). *A sequence of $n \geq 1$ of funnels $[fnl_i]_{i=0}^{n-1}$ over symbols V is a funnel-loop iff the following hold.*

FL.1 *The destination region of a funnel is included in the source region of the following funnel.*

$$\forall 0 \leq i < n-1, V : D_i(V) \rightarrow S_{i+1}(V)$$

FL.2 *The destination region of the last funnel D_{n-1} is contained in the source region of the first funnel S_0.*

$$\forall V : D_{n-1}(V) \rightarrow S_0(V)$$

We define the paths through a funnel-loop $floop$, $\mathcal{L}(floop)$, as the infinite paths obtained by infinite concatenation of the paths of the funnels in the corresponding chain and write $floop \models \phi$ meaning that ϕ holds in all such paths. For every funnel different from the last one, Hyp. FL.1 ensures that we can extend every path of such funnel, ending in its destination region, by following the transition relation of the next funnel. Therefore, every path starting in any source region will eventually reach the destination region of the last funnel:

$$floop \models (\bigvee_{i=0}^{n-1} S_i) \ \mathbf{U} \ D_{n-1}$$

By Hyp. FL.2 every time we reach the destination region of the last funnel associated with $floop$ we are also in the source region of the first funnel. Therefore, we can extend the execution by appending another finite number of steps: a finite path starting from S_0 and ending in the last destination region D_{n-1}. We can do this infinitely many times obtaining infinite paths.

$$floop \models \mathbf{G}((\bigvee_{i=0}^{n-1} S_i) \ \mathbf{U} \ D_{n-1})$$

We propose to identify a non-empty set of fair paths for a transition system M as a funnel-loop $floop$; every path through $floop$ must correspond to an infinite fair execution of M. The totality of the transition relation of each funnel (F.1) and their chaining (FL.1, FL.2) ensure that all the paths in $\mathcal{L}(floop)$ are infinite. We need such paths to be fair paths, hence they must visit the fairness condition infinitely often. By construction of $floop$ we know that every path goes through each S_i and each D_i infinitely many times. Since by FL.1 and FL.2 for every source region S_i, there exists a destination region D_j that is contained in it, it is sufficient to require one of the destination regions to contain only fair states. Without loss of generality we assume such a region to be the last one. These conditions ensure that $floop$ represents a set of fair paths of M. However, such set might be empty or non-reachable in M. Therefore, we finally require the union of the source regions to contain at least one state reachable in M. The existence of such state is sufficient to conclude non-emptiness of $\mathcal{L}(floop)$ because the transition relation of each funnel always allows for a successor state (F.1) and, by induction, this ensures that every region and the language of $floop$ are not empty. Th 1 shows that these requirements are sufficient for a funnel-loop to prove the existence of a fair path in M.

Theorem 1. *Let $M \doteq \langle V, I_M, T_M, F_M \rangle$ be a fair transition system. Let $floop$ be a funnel-loop of length n over the symbols V and funnels $[fnl_i]_{i=0}^{n-1}$ such that:*

FF.1 *There is at least one state reachable in M in the union of the source regions of $floop$:*

$$M \not\models \mathbf{G} \neg \bigvee_{i=0}^{n-1} S_i$$

FF.2 *The destination region of the last funnel must contain only fair states of M.*

$$\forall V \ : \ D_{n-1}(V) \rightarrow F_M(V)$$

FF.3 *Every transition of every funnel underapproximates the transition relation of M. For every funnel fnl_i in $[fnl_i]_{i=0}^{n-1}$:*

$$\forall V, V' \ : \ S_i(V) \wedge T_i(V, V') \rightarrow T_M(V, V')$$

Then M admits at least one fair path.

4 Automated synthesis of funnel loops

This section describes our approach to automate the synthesis of a funnel loop. Alg. 1 describes the main steps of the procedure. We reduce the synthesis problem to a sequence of SMT queries. In order to reduce the search space, we only look for deterministic funnel loops by requiring that each transition relation of each funnel is deterministic. More in detail, Alg. 1 enumerates candidate conjunctive fair loops of the fair transition system and, for each loop, it generates a sequence of parameterised candidate funnel loops. The procedure then tries to find an assignment to the parameters such that the candidate funnel loop meets all the hypotheses of Defs. 1 and 2 and of Th. 1.

In the following we consider parametric expressions that are linear combinations of the variables of the system, i.e. $\sum_{v_i \in V} \lambda_i \cdot v_i$, where λ_i are the parameters.

Algorithm 1 SEARCH-FUNNEL(M)

```
1:  for $\langle v_0, abst\_s, abst\_t \rangle \in$ GENERATE-ABSTRACT-LOOPS($M$) do
2:    for $fnl\_template \in$ GENERATE-TEMPLATES($v_0, abst\_s, abst\_t$) do
3:      $ef\_constrs \leftarrow fnl\_template.ef\_constraints()$
4:      $\langle found, model \rangle \leftarrow$ SEACH-PARAMETER-ASSIGNMENT($ef\_constrs$)
5:      if $found == \top$ then
6:        return $\langle model, fnl\_template \rangle$
7:      end if
8:    end for
9:  end for
10: return $unknown$
```

We use a method called NEW-PARAMETRIC-EXPR to generate such linear combinations of symbols and parameters, and we refer to the set of all parameters as P.

The procedure relies on ranking functions to perform 2 different tasks. Alg. 2 tries to synthesise ranking functions to avoid considering candidate abstract loops for which we know a ranking function exists. The existence of the ranking function proves that the loop must eventually terminate, hence it cannot correspond to an infinite path. Then, ranking function templates are also used as components for the funnels of the funnel-loop template generated by Alg. 3. In both cases as template for the ranking functions we consider the PR-ranking template described in [31].

We first describe how we represent and enumerate candidate abstract loops for the transition system M. Then, we describe how a funnel-loop template is generated from a candidate abstract loop and the search problem associated with a funnel-loop template. Finally, we describe the approach we adopt to perform the search.

Given a fair transition system $M \doteq \langle V, I_M, T_M, F_M \rangle$ we describe a candidate conjunctive fair abstract loop of length n for M as a sequence of abstract states $abst\_s \doteq [abst\_s_i(V)]_{i=0}^{n-1}$, transitions $abst\_t \doteq [abst\_t_i(V, V')]_{i=0}^{n-2}$ and an initial state $\boldsymbol{v}_0$ such that: (i) $\boldsymbol{v}_0 \models abst\_s_0(V)$, (ii) $\boldsymbol{v}_0$ is reachable in M, (iii) one of the abstract states underapproximates the fair states, and (iv) the abstract path is an implicant for a path of the same length in M:

$$\begin{aligned}&\forall V_0, \ldots, V_{n-1} : (\bigwedge_{i=0}^{n-1} abst\_s_i(V_i) \wedge \bigwedge_{i=0}^{n-2} abst\_t_i(V_i, V_{i+1})) \to \bigwedge_{i=0}^{n-2} T_M(V_i, V_{i+1})\\ &\exists i \; \forall V : abst\_s_i(V) \to F_M(V).\end{aligned}$$

Both the abstract states and the abstract transitions are built as formulae over a finite set of predicates. Without loss of generality, and to simplify the presentation, we assume the fair abstract state to be the first one. The enumeration of abstract loops is performed by Alg. 2. The procedure is based on Bounded

Algorithm 2 GENERATE-ABSTRACT-LOOPS(M)

```
1: $\langle V, I, T, bad \rangle \leftarrow$ ENCODE-BMC-FAIR-ABSTRACT-LOOP$(M)$
2: for $k \in [0, 1, 2, \ldots]$ do
3:     $query \leftarrow I(V_0) \wedge \bigwedge_{i=0}^{k-1} T(V_i, V_{i+1}) \wedge bad(V_k)$
4:     $\langle sat, model \rangle \leftarrow$ SMT-SOLVE$(query)$
5:     $refs \leftarrow []$
6:     while sat do
7:         $\langle abst\_s, abst\_t \rangle \leftarrow$ GET-IMPLICANT$(model, query)$
8:         $\langle is\_ranked, rf \rangle \leftarrow$ RANK-LOOP$(abst\_s, abst\_t)$
9:         if $is\_ranked$ then
10:            $\langle V, I, T, bad \rangle \leftarrow$ REMOVE-RANKED-LOOPS$(V, I, T, bad, rf)$
11:        else
12:            $v_0 \leftarrow$ GET-LOOPBACK-STATE$(model)$
13:            yield $\langle v_0, abst\_s, abst\_t \rangle$
14:            $refs.append(\neg(\bigwedge_{s \in abst\_s} s \wedge \bigwedge_{t \in abst\_t} t))$
15:        end if
16:        $query \leftarrow I(V_0) \wedge \bigwedge_{i=0}^{k-1} T(V_i, V_{i+1}) \wedge bad(V_k) \wedge \bigwedge_{ref \in refs} ref$
17:        $\langle sat, model \rangle \leftarrow$ SMT-SOLVE$(query)$
18:    end while
19: end for
```

Model Checking (BMC) [3], for the enumeration of candidate paths, and on the computation of an implicant for each path.

Line 1 performs the usual BMC encoding for the search of a fair loop, where the loop-back state is identified in the abstract space defined by the predicates in the transition relation and fairness condition of M. The last state and the loop-back state must agree on the truth assignment of all the predicates in the transition relation and fairness condition, hence they may not be the very same assignment. We then rely on a SMT-solver to identify fair lasso paths of increasing length k, as done for the abstract liveness-to-safety algorithm of [14]. Then, at line 8 we first try to synthesise a ranking function for such abstract loop. The method RANK-LOOP implements the procedure described in [31] for PR-ranking templates. If we succeed in identifying a ranking function, we refine our transition system such that we avoid enumerating other loops ranked by the same function, as described in [14] (REMOVE-RANKED-LOOPS, line 10). Otherwise, from the path we extract the assignment to the loop-back state and return it together with the current abstract path. If no abstract loop of length k exists, we clear the list of refinements and enumerate the candidate loops of length $k + 1$.

Alg. 3 shows the procedure we use to generate a funnel-loop template from a candidate abstract loop. We generate a funnel-loop of the same length as the abstract loop. Line 1 selects a list of natural numbers to be used to generate the funnel-loop templates. Each number corresponds to the amount of parametric inequalities added to each abstract state to define the corresponding source region

Algorithm 3 GENERATE-TEMPLATES($\boldsymbol{v}_0, abst\_s, abst\_t$)

```
1: $ineqs \leftarrow$ HEURISTIC-PICK-NUM-INEQS($abst\_s, abst\_t$)
2: for $ineq \in ineqs$ do
3:     $n \leftarrow len(abst\_s)$
4:     $funnels \leftarrow []$
5:     for $i \in [0..n-2]$ do
6:         $src \leftarrow abst\_s[i] \wedge \bigwedge_{j=0}^{ineq-1}$ NEW-PARAMETRIC-EXPR$(V) \geq 0$
7:         $rf \leftarrow$ NEW-PARAMETRIC-EXPR$(V)$
8:         $t \leftarrow \top$
9:         for $v_{i+1} \in V_{i+1}$ do
10:            if $v_{i+1} = f(V_i) \in abst\_t[i]$ for some function $f$ then
11:                $t \leftarrow t \wedge v_{i+1} = f(V_i)$
12:            else
13:                $t \leftarrow t \wedge v_{i+1} =$ NEW-PARAMETRIC-EXPR$(V_i)$
14:            end if
15:        end for
16:        $dst(V) \leftarrow \exists V_0 \; : \; src(V_0) \wedge rf(V_0) = \mathbf{0} \wedge t(V_0, S)$
17:        $funnels.append(Funnel(src, t, rf, dst))$
18:    end for
19:    yield FUNNEL-LOOP($funnels, \boldsymbol{v}_0$)
20: end for
```

of a funnel template (line 6). The higher the number the more freedom will the template have in shrinking the regions, but in the search problem we will have more parameters and a larger space to explore. Notice that, since by construction of the abstract loop one of the *abst_s* is fair, then also the corresponding destination region in the funnel-loop template will be fair. We create the funnel template corresponding to the i^{th} abstract state $abst\_s[i]$ and transition $abst\_t[i]$ in lines 5–18. We define the transition relation t of the funnel as a deterministic functional assignment as follows. For each symbol $v_{i+1} \in V_{i+1}$, if $abst\_t_i$ already contains a functional assignment for v_{i+1}, then we use that (line 11). Otherwise, we generate a functional assignment for v_{i+1} as a parametric expression over the symbols in V (line 13). We define the destination region of a funnel implicitly as the set of states reachable in one step from $S(V) \wedge \mathrm{RF}(V) = \mathbf{0}$ (line 16). Finally, the procedure returns the funnel-loop template associated with the list of parametric funnels and initial state $\boldsymbol{v}_0$.

We now describe the $\exists\forall$ quantified formula that corresponds to the synthesis problem of a funnel-loop template and the procedure we use to solve it. Every instance of the funnel-loop template must satisfy all hypotheses of Defs. 1, and 2 and of Th. 1. In the hypotheses, for every funnel $fnl_i \doteq \langle S_i, T_i, D_i, \mathrm{RF}_i \rangle$, we replace each destination region D_i with the quantified formula:

$$\exists V_0 : S_i(V_0) \wedge \mathrm{RF}_i(V_0) = \mathbf{0} \wedge T_i(V_0, V). \tag{1}$$

Every instance of the funnel-loop template must contain a fair region since $abst\_s_0$ is a subset of the fair states and S_0, by construction, underapproximates $abst\_s_0$. We ensure that Hyp. FR.1 holds by requiring that $\boldsymbol{v}_0$ is in the source region of the first funnel fnl_0 with the constraint:

$$\exists P \ : \ S_0(\boldsymbol{v}_0, P). \tag{2}$$

Hyp. F.1 holds by construction since each transition relation T_i of every funnel template fnl_i is a functional assignment without any circular dependency. Hyp. F.4 holds since we implicitly defined the destination region of each funnel fnl_i as the set of states reachable in one step from $S_i \wedge \mathrm{RF}_i = \mathbf{0}$. Then, we ensure that every instantiation of every funnel template fnl_i in the funnel-loop template satisfies hypotheses F.2 and F.3 by requiring that the following hold:

$$\exists P \ \forall V, V' : (S_i(V, P) \wedge \mathrm{RF}_i(V, P) > \mathbf{0} \wedge T_i(V, V', P)) \rightarrow S_i(V', P) \tag{3}$$

$$\exists P \ \forall V, V' : (S_i(V, P) \wedge \mathrm{RF}_i(V, P) > \mathbf{0} \wedge T_i(V, V', P)) \rightarrow \mathrm{RF}_i(V, P) > \mathrm{RF}_i(V', P) \tag{4}$$

The funnels must be correctly chained for Hyp. FL.1 to hold. For this reason we require every two consecutive funnel templates fnl_i and fnl_{i+1} in the funnel-loop template to satisfy the following:

$$\exists P \ \forall V, V' : (S_i(V, P) \wedge \mathrm{RF}_i(V, P) = \mathbf{0} \wedge T_i(V, V', P)) \rightarrow S_{i+1}(V', P) \tag{5}$$

Similarly, considering the first and last funnels fnl_0 and fnl_{n-1}, for Hyp. FL.2 we require:

$$\exists P \ \forall V, V' : (S_{n-1}(V, P) \wedge \mathrm{RF}_{n-1}(V, P) = \mathbf{0} \wedge T_{n-1}(V, V', P)) \rightarrow S_0(V', P) \tag{6}$$

This ensures that D_{n-1} is a subset of S_0. We have observed above that S_0 contains only fair states, hence FR.2 holds. Finally, we require each funnel-loop instance to underapproximate M (Hyp. FR.3) by requiring the following to hold for every funnel fnl:

$$\exists P \ \forall V, V' \ : \ S(V, P) \wedge T(V, V', P) \rightarrow T_M(V, V'). \tag{7}$$

The final synthesis problem is then given by the conjunction of all the constraints (1)–(7). In order to solve it, we apply a combination of the EF-SMT procedure of [16] and the application of Motzkin's transposition theorem [33] to reduce the problem into a purely existentially-quantified one which can then be solved via standard quantifier-free SMT reasoning: we first try to apply EF-SMT, and resort to the elimination of universal quantifiers only if this fails to provide a definite answer.

5 Related work

Most of the literature in verification of temporal properties of infinite-state transition systems, hybrid automata and termination analysis focuses on the universal case, while the existential one has received relatively little attention. The most closely related work is [6]. The key difference is that the procedure we presented in [6] is partly interactive, while the approach presented here is fully automatic. Furthermore, there is a difference at the technical level in the way the approaches partition the problem. In [6], the idea is to synthesise a partitioned structure called $\mathcal{R}$-abstraction out of a set of components, called $\mathcal{AG}$-skeletons. Each component is obtained by considering only a subset of the symbols of the system, and is used to describe a set of infinite paths for such symbols. Here, instead, we act on the monolithic system, but partition the fair path into funnels.

Also related are the works concerned with proving *program non-termination.* [21] and [11] are based on the notion of closed recurrence set, that corresponds to proving the non-termination of a relation. [5] and [30] search for non-terminating executions via a sequence of safety queries. Other approaches look for specific classes of programs ([18] and [24] prove the decidability of termination for linear loops over the integers), or specific non-termination arguments (in [32] non-termination is seen as the sum of geometric series). However, none of these works deals with fairness and they rely on the existence of a control flow graph, whereas we work at the level of transition system.

[13] reduces the verification of the universal fragment of CTL on a infinite-state transition system to the problem of deciding whether a program always returns true. The approach can be applied also on LTL properties by relying on a reduction based on prophecy variables and it relies on some off-the-shelf tool for the analysis of the program. Therefore, its capability of proving or identifying a counterexample for some property depends on the ones of the considered underlying tool.

[12] explicitly deals with fairness for infinite-state programs supporting full CTL*: it is able to deal with existential properties and to provide fair paths as witnesses. The approach focuses on programs manipulating integer variables, with an explicit control-flow graph, rather than more general symbolic transition systems expressed over different theories (including real arithmetic). Another approach supporting full CTL* is proposed in [25]. The work presents a model checking algorithm for the verification of CTL* on finite-state systems and a deductive proof system for CTL* on infinite-state systems. In the first case they reduce the verification of CTL* properties to the verification of properties without temporal operators and a single fair path quantifier in front of the formula. To the best of our knowledge there is no generalisation of this algorithm, first reported in [26] and then also in [27], to the infinite-state setting. The rules presented in the second case have been exploited in [2] to implement a procedure for the verification of CTL properties, while our objective is the falsification of LTL properties. Moreover, in these settings ([12], [25]) there is no notion of non-zenoness.

The works on *timed automata* are less relevant: although the concrete system may exhibit no lasso-shaped witnesses, due to the divergence of clocks, the problem is decidable, and lasso-shaped counterexamples exist in finite bi-simulating abstractions. This view is adopted, for example, in UPPAAL [1]. Other tools directly search for non lasso-shaped counterexamples, but the proposed techniques are specific for the setting of timed automata [7, 28] and lack the generality of the method proposed in this paper. Finally, our approach can be applied also to *hybrid systems*. However, the implementation relies on an approximation of the nonlinearities which, from our experiments, appears too coarse for this context.

6 Experimental evaluation

Implementation. We have implemented these procedures in a prototype, called F3[3] (for FindFairFunnel), written in Python. F3 uses MathSAT5 [9] and Z3 [34] as underlying SMT engines, interacting with them through pysmt [19]. F3 takes as input a transition system M and a fairness condition F, and tries to identify a funnel that proves that M admits at least 1 path that visits F infinitely-often. We then employ the usual tableau construction to support LTL specifications via reduction to the previous case. In order to support timed systems, we use the product construction described in [8] to remove all zeno-paths of the model. F3 enumerates funnel templates in increasing order of complexity. By default, F3, considers a minimum of 0 and a maximum of 2 inequalities in the implementation of heuristic-pick-num-ineqs of Alg. 3. An important optimization is that F3 generates ranking function templates (line 6 of Alg. 3) only when it finds a pair of abstract states that prescribe the same assignment to the boolean variables of M; if the abstract states differ in their boolean variables, rf is simply set to the constant $\mathbf{0}$. This avoids the introduction of unnecessary parameters for funnels which do not need an explicit ranking function. Finally, when applying the Motzkin's transposition theorem to solve the parameter synthesis problems, F3 replaces non-linear terms with fresh symbols, in order to obtain a linear system. This simple way of handling non-linearities has the benefit of being very easy to implement; on the other hand, however, it can produce very coarse approximations, which can prevent F3 from finding counterexamples in cases where non-linearities play a significant role.

Benchmarks. In order to evaluate the effectiveness of our method, we have evaluated F3 on a wide range of benchmarks coming from different domains, from software (non)termination to timed automata and infinite-state symbolic transition systems. More specifically, we considered a total of 455 benchmarks, divided into 6 categories:

LS consists of 52 nonterminating linear software benchmarks taken from the C programs of the software termination competition;

NS contains 30 nonlinear software programs, of which 29 have been taken from [11] and one from [6];

ITS are 70 LTL falsification problems on infinite-state systems; 2 of such problems are proof obligations generated in the verification of a contract-based

[3] the tool and the benchmarks can be downloaded from `https://github.com/EnricoMagnago/F3`

design, 29 come from the scaling to up to 30 processes of a model of the bakery mutual exclusion protocol in which we introduced a bug, other 29 come from the scaling to up to 30 processes of a semaphore-based synchronisation protocol, and the last 10 are instances we created;

TA contains 174 LTL falsification problems on timed automata; we consider 6 different protocols taken from [17] (*critical*, *csma*, *fddi*, *fischer*, *lynch* and *train*) and scale each of them from 1 to 30 processes;

TTS consists of 120 LTL falsification problems on timed transition systems, of which 116 come from the scaling from 1 to 30 processes of 4 protocols (inspired by the *csma*, *fischer*, *lynch* and *token ring* protocols), and 4 are handcrafted instances;

HS are 9 LTL falsification problems on hybrid systems (encoded as nonlinear infinite-state transition systems) taken from [6].

F3 only handles symbolic transition systems, and not software programs; therefore, we have encoded the software benchmarks as infinite-state transition systems by introducing an explicit program counter as state variable. Moreover, since F3 only supports systems with boolean, integer and real variables, we have not considered programs that involve recursion or dynamic memory allocation.

Competitor tools. We compare F3 with the following state-of-the-art tools: Anant [11], AProVe [20], DiVinE3 [22], MITLBMC [29], nuXmv [7], T2 [4], Ultimate [23] and Uppaal [15]. Most of the other tools are however not able to handle all the bechmarks we have considered. Therefore, we limit their application as follows:

- we ran Anant, AProVe and T2 only on the software nontermination problems (LS and NS groups);
- we ran DiVinE3, MITLBMC and Uppaal only on the time automata (TA) benchmarks; moreover, since Uppaal supports only a fragment of LTL which is not sufficient to express the properties of the *fischer* and *lynch* benchmarks, we could run it only on 116 of the 174 TA instances;
- as Ultimate doesn't support non-linear arithmetic, we didn't run it on the NA family. Moreover, since it supports LTL specifications but works on programs rather than transition systems, we translated the benchmarks to LTL verification problems on software programs, using the same approach described in [14].
- nuXmv is the only other tool (besides F3) that supports all the benchmarks. Since our focus is falsification of universal properties (or dually verification of existential ones), we ran nuXmv using only its BMC engine.

Results. We performed our experiments on a machine running Ubuntu 20.04 equipped with an Intel(R) Xeon(R) Gold 6226R 2.90GHz CPU, using a 1h timeout and a memory limit of 30 GB for each benchmark. A summary of the evaluation results is reported in Table 1. The table shows, for each tool, the number

Table 1. Summary of experimental results (number of solved instances per benchmark family).

Benchmark family	**F3**	**Anant**	**AProVe**	**DiVinE3**	**MITLBMC**	**nuXmv**	**T2**	**Ultimate**	**Uppaal**
LS (52)	**52**	38	43	–	–	28	38	49	–
NS (30)	**29**	25	5	–	–	14	2	–	–
ITS (70)	**57**	–	–	–	–	4	–	8	–
TA (174)	137	–	–	43	**151**	90	–	0	103
TTS (120)	**55**	–	–	–	–	8	–	1	–
HS (9)	0	–	–	–	–	0	–	–	–
Total (455)	**330**	63	48	43	151	144	40	58	103

Entries marked with "–" denote that the tool cannot handle the given benchmarks.

of solved instances in each benchmark family. When a tool is not applicable to a specific family, this is marked with "-". (More detailed information on the comparison of F3 with the other tools on individual benchmark families is available in Appendix A.) From the table, we can see that F3 not only solved the highest number (by far) of instances overall, but it is also the tool that solved the highest number of instances in all categories with the exception of timed automata. In this category F3 is outperformed only by MITLBMC, which implements a technique explicitly developed for timed automata. This demonstrates the generality of our approach, although (unsurprisingly) it is possible to define more efficient procedures to target specific classes of problems. On the software benchmarks (linear and non-linear) F3 fails to provide an answer in only 1 case (the nonlinear one taken from[6]). Therefore, while being coarse-grained, the approximation of the nonlinear terms used by F3 appears to be sufficient in these cases. However, the hybrid benchmarks highlight the limitations of such approximation. In fact, F3 was unable to provide an answer in all 9 cases. These instances can be solved successfully with the approach of [6], which however requires user guidance and is therefore not fully automatic. In fact, we are not aware of any automatic tool that is able to solve them. Finally, we should remark that unlike F3 several of the competitor tools (with the exception of MITLBMC and nuXmv in BMC mode) are also able to prove that a universal property holds, whereas F3 can only find counterexamples. On the other hand, however, our techniques can be easily integrated with approaches focusing on proving properties, such as [8,14].

7 Conclusions and future work

In this paper we presented an automated approach to the verification of existential properties for infinite-state systems. We adopt an approach to build an implicit presentation of fair paths, that may not have a lasso-shape structure, using an abstract representation of the trace in form of a sequence of funnels. The approach alternates between finding candidate counterexample skeleta in the abstract space, and proving whether they admit a concretization.

The experimental evaluation, carried out on a wide set of benchmarks, demonstrates that the approach is very effective, being able to solve realistic benchmarks from many different domains, and also general, being competitive with other specialized tools.

In the future, we plan to integrate the partitioning techniques presented in [6] in an automated setting, and to explore the possibility of hierarchically decomposed proofs.

References

1. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: SFM-RT, vol. 3185 of LNCS, Springer (2004).
2. Beyene, T.A., Popeea, C., Rybalchenko, A.: Solving existentially quantified horn clauses. In: CAV, vol. 8044 of LNCS, Springer (2013).
3. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. In Adv. Comput. **58** (2003).
4. Brockschmidt, M., Cook, B., Ishtiaq, S., Khlaaf, H., Piterman, N.: T2: temporal property verification. In: TACAS, vol. 9636 of LNCS, Springer (2016).
5. Chen, H.Y., Cook, B., Fuhs, C., Nimkar, K., O'Hearn, P.W.: Proving nontermination via safety. In: TACAS, vol. 8413 of LNCS, Springer (2014).
6. Cimatti, A., Griggio, A., Magnago, E.: Proving the existence of fair paths in infinite-state systems. In: VMCAI, vol. 12597 of LNCS, Springer (2021).
7. Cimatti, A., Griggio, A., Magnago, E., Roveri, M., Tonetta, S.: Extending nuxmv with timed transition systems and timed temporal properties. In: CAV, vol. 11561 of LNCS, Springer (2019).
8. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: Verifying LTL properties of hybrid systems with k-liveness. In: CAV, vol. 8559 of LNCS, Springer (2014).
9. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The mathsat5 SMT solver. In: TACAS, vol. 7795 of LNCS, Springer (2013).
10. Clarke, E.M., Grumberg, O., Hamaguchi, K.: Another look at LTL model checking. Formal Methods in System Design **10**(1) (1997)
11. Cook, B., Fuhs, C., Nimkar, K., O'Hearn, P.W.: Disproving termination with overapproximation. In: FMCAD, IEEE (2014).
12. Cook, B., Khlaaf, H., Piterman, N.: Verifying increasingly expressive temporal logics for infinite-state systems. J. ACM **64**(2) (2017).
13. Cook, B., Koskinen, E., Vardi, M.Y.: Temporal property verification as a program analysis task - extended version. Formal Methods Syst. Des. **41**(1), 66–82 (2012).
14. Daniel, J., Cimatti, A., Griggio, A., Tonetta, S., Mover, S.: Infinite-state liveness-to-safety via implicit abstraction and well-founded relations. In: CAV, vol. 9779 of LNCS, Springer (2016).
15. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B.: Uppaal smc tutorial. International Journal on Software Tools for Technology Transfer **17**(4), (2015).
16. Dutertre, B.: Solving exists/forall problems with yices. In: SMT Workshop (2015)
17. Farkas, R., Bergmann, G.: Towards reliable benchmarks of timed automata. In: Proceedings of the 25th PhD Mini-Symposium (2018)

18. Frohn, F., Giesl, J.: Termination of triangular integer loops is decidable. In: CAV, vol. 11562 of LNCS, Springer (2019).
19. Gario, M., Micheli, A.: Pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms. In: SMT Workshop (2015)
20. Giesl, J., Brockschmidt, M., Emmes, F., Frohn, F., Fuhs, C., Otto, C., Plücker, M., Schneider-Kamp, P., Ströder, T., Swiderski, S., Thiemann, R.: Proving termination of programs automatically with aprove. In: IJCAR, vol. 8562 of LNCS, Springer (2014).
21. Gupta, A., Henzinger, T.A., Majumdar, R., Rybalchenko, A., Xu, R.: Proving non-termination. In: POPL, ACM (2008).
22. Havlíček, J.: Untimed LTL Model Checking of Timed Automata. Ph.D. thesis. Masaryk University, (2013).
23. Heizmann, M., Christ, J., Dietsch, D., Ermis, E., Hoenicke, J., Lindenmann, M., Nutz, A., Schilling, C., Podelski, A.: Ultimate automizer with smtinterpol - (competition contribution). In: TACAS, vol. 7795 of LNCS, Springer (2013).
24. Hosseini, M., Ouaknine, J., Worrell, J.: Termination of linear loops over the integers. In: ICALP, vol. 132 of LIPIcs, (2019).
25. Kesten, Y., Pnueli, A.: A compositional approach to CTL* verification. Theor. Comput. Sci. **331**(2-3), (2005).
26. Kesten, Y., Pnueli, A., Raviv, L.: Algorithmic verification of linear temporal logic specifications. In: ICALP, vol. 1443 of LNCS, Springer (1998).
27. Kesten, Y., Pnueli, A., Raviv, L., Shahar, E.: Model checking with strong fairness. Formal Methods Syst. Des. **28**(1), (2006).
28. Kindermann, R., Junttila, T.A., Niemelä, I.: Beyond lassos: Complete smt-based bounded model checking for timed automata. In: FORTE, vol. 7273 of LNCS, Springer (2012).
29. Kindermann, R., Junttila, T.A., Niemelä, I.: Bounded model checking of an MITL fragment for timed automata. In: ACSD, IEEE Computer Society (2013).
30. Larraz, D., Nimkar, K., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: Proving non-termination using max-smt. In: CAV, vol. 8559 of LNCS, Springer (2014).
31. Leike, J., Heizmann, M.: Ranking templates for linear loops. Log. Methods Comput. Sci. **11**(1) (2015).
32. Leike, J., Heizmann, M.: Geometric nontermination arguments. In: TACAS, vol. 10806 of LNCS, Springer (2018).
33. Motzkin, T.S.: Two consequences of the transposition theorem on linear inequalities. Econometrica (pre-1986) **19**(2), 184 (1951)
34. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: TACAS, vol. 4963 of LNCS, Springer (2008).
35. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science. IEEE Computer Society (1977).
36. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Banff Higher Order Workshop, vol. 1043 of LNCS, Springer (1995)

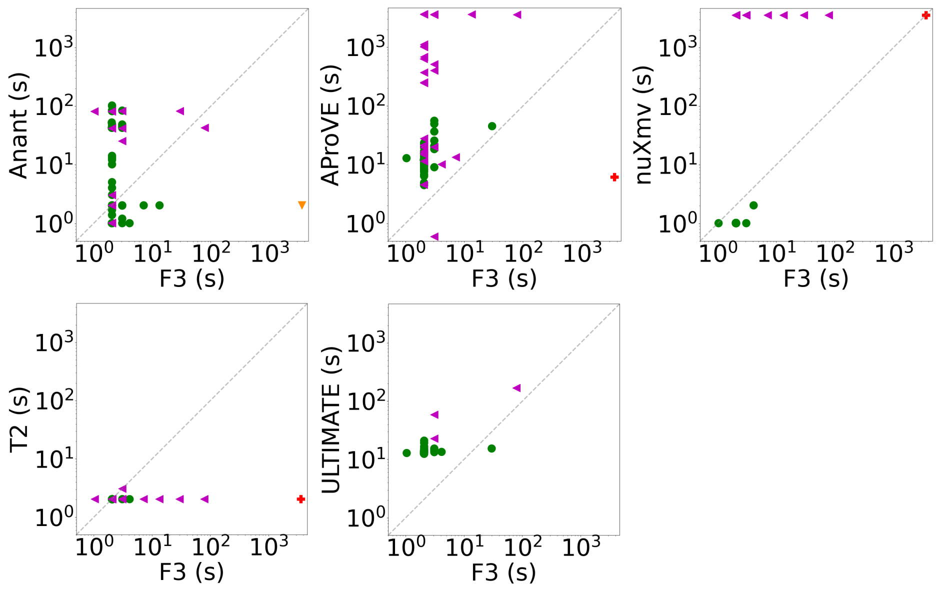

Fig. 2. Execution time of F3 compared to ANANT, APROVE, NUXMV, T2 and ULTIMATE on software benchmarks.

A Experimental evaluation plots

In the plots we use • to indicate instances in which both tools provided an answer, ◀ [resp. ▼] to indicate instances in which the other tool [resp. F3] did not provide an answer and + for instances in which both tools failed to provide an answer.

The plots in Fig. 2 compare F3 with ANANT [11], APROVE [20], T2 [4] and NUXMV [7] on linear and non-linear software benchmarks and with ULTIMATE [23] only on the linear benchmarks since ULTIMATE does not support non-linear expressions. We consider 52 linear software benchmarks taken from the software termination competition and 30 nonlinear software benchmarks, 29 of which taken from [11] and one taken from [6]. We encode each of them as a infinite state transition system for F3 with an explicit program counter as state variable. Notice that in doing this we are loosing the structure of the program. F3 handles only transition systems with boolean, integer and real variables, for this reason we have not considered software benchmarks that involve recursion or dynamic memory allocation. F3 provides an answer in all cases but on the nonlinear instance we introduced in [6].

Fig. 3 reports the comparison of F3 with ULTIMATE and NUXMV on 70 LTL falsification problems on infinite state systems. 2 of such problems are proof obligations generated in the verification of a contract-based design, 29 come from the scaling to up to 30 processes of a model of the bakery mutual exclusion protocol in which we introduced a bug, other 29 come from the scaling to up to

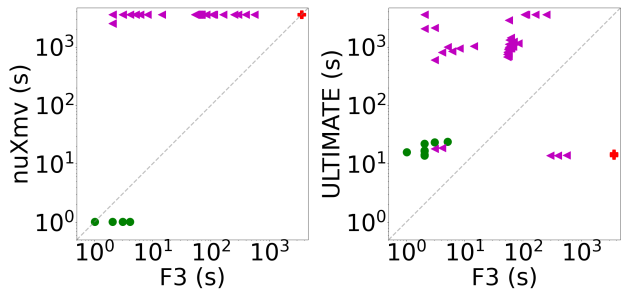

Fig. 3. Execution time of F3 compared to nuXmv and Ultimate on infinite state transition systems.

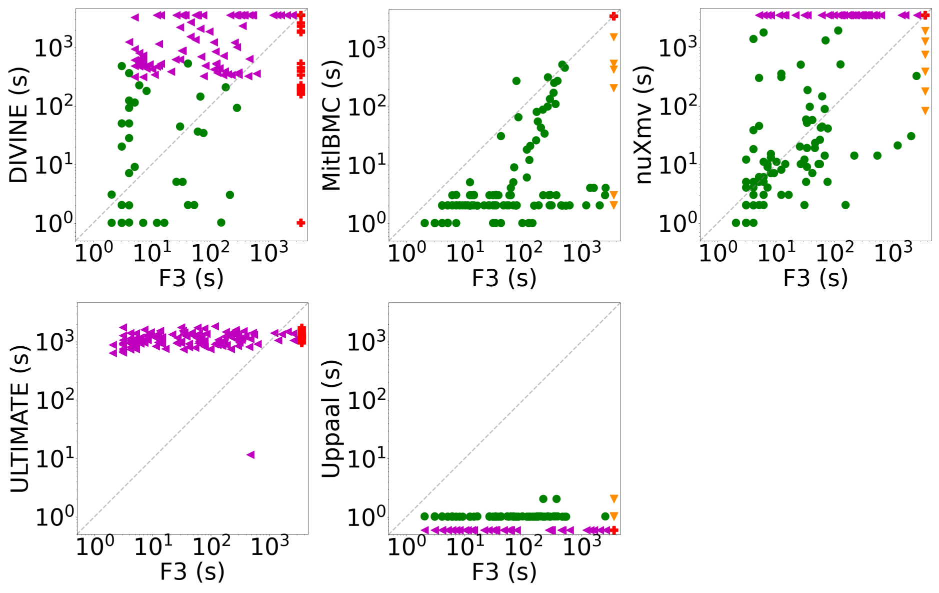

Fig. 4. Execution time of F3 compared to DiVinE3, MITLBMC, nuXmv, Ultimate and Uppaal on timed automata.

30 processes of a semaphore based synchronisation protocol and the last 10 are instances we created.

In Fig. 4 we compare F3 with DiVinE3, MITLBMC, nuXmv, Ultimate and Uppaal on LTL falsification problems on timed automata. We consider 174 instances, coming from the scaling from 1 to 30 processes of 6 different protocols taken from [17]: *critical*, *csma*, *fddi*, *fischer*, *lynch* and *train*. Uppaal supports only a fragment of LTL and we could not express in such fragment the properties of the *fischer* and *lynch* examples, all such instances are marked as unknowns in the plots.

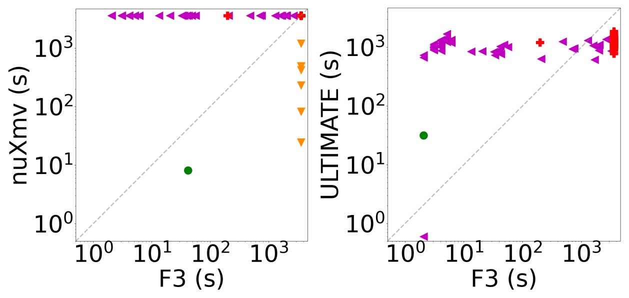

Fig. 5. Execution time of F3 compared to nuXmv and Ultimate on timed transition systems.

We compare F3 with Ultimate and nuXmv on the falsification of LTL specifications for timed transition systems in Fig. 5. We consider 120 instances, 116 of which come from the scaling from 1 to 30 processes of 4 protocols and 4 additional instances we created. The scaling models are inspired by the *csma*, *fischer*, *lynch* and *token ring* protocols.

We compare F3 with nuXmv on 9 instances of LTL falsification of hybrid systems taken from [6]. The hybrid systems are encoded as nonlinear infinite state transition systems and F3 fails to provide an answer in all such cases. While the approximation of non-linear terms was sufficient for most of the nonlinear software benchmarks it appears to be too coarse in these cases.

B Theorems and proofs

B.1 Proof of Th. 1

Theorem *Let $M \doteq \langle V, I_M, T_M, F_M \rangle$ be a fair transition system. Let $floop$ be a funnel-loop of length n over the symbols V and funnels $[fnl_i]_{i=0}^{n-1}$ such that:*

FF.1 *There is at least one state reachable in M in the union of the source regions of $floop$:*

$$M \not\models \mathbf{G} \neg \bigvee_{i=0}^{n-1} S_i$$

FF.2 *The destination region of the last funnel must contain only fair states of M.*

$$\forall V \ : \ D_{n-1}(V) \rightarrow F_M(V)$$

FF.3 *Every transition of every funnel underapproximates the transition relation of M. For every funnel fnl_i in $[fnl_i]_{i=0}^{n-1}$:*

$$\forall V, V' \ : \ S_i(V) \wedge T_i(V, V') \rightarrow T_M(V, V')$$

Then admits at least one fair path.

Proof. We first prove that every path in $\mathcal{L}(floop)$ is infinite. Then we prove that every such path is fair with respect to the fairness condition F_M and that every step in every such path satisfies the transition relation T_M. Finally, we prove that $\mathcal{L}(floop)$ allows for at least one path which is a suffix of some path of M.

- Every path in $\mathcal{L}(floop)$ is infinite. Consider a funnel $fnl \doteq \langle S, T, D, \mathrm{R\scriptstyle F} \rangle$ in $floop$. Hyp. F.1 ensures that its transition relation T allows for a successor state for every state in S. Hyp. F.2 ensures that every path of fnl remains in S while $\mathrm{R\scriptstyle F} > \mathbf{0}$. Hyp. F.3 ensures that every such path will eventually reach a state in $S \wedge \mathrm{R\scriptstyle F} = \mathbf{0}$. Hyp. F.4 ensures that every state in such region in one T step reaches a state in D. Therefore, every path starting from the source region S of each funnel can be extended until it reaches its destination region D. If fnl_{i-1} has a successor fnl_i in $floop$, by Hyp. FL.1 the destination region D_{i-1} is included in S_i: every state in D_{i-1} is also in S_i. Therefore, the concatenation of fnl_{i-1} and fnl_i allows to extend every path starting from either S_{i-1} or S_i until it reaches D_i. By induction this shows that the funnel chain allows the extension of every path starting from the union of the source regions until it reaches the last destination region:
$$floop \models (\bigvee_{i=0}^{n-1} S_i) \; U \; D_{n-1}$$
Hyp. FL.2 requires the last destination region D_{n-1} to be a subset of the first source region S_0. As stated above, we can extend every path starting in every region until it reaches D_{n-1}, hence from S_0 we reach D_{n-1} again in a finite number of steps and at least one. Therefore, since we can extend each path of a finite non-zero number of steps infinitely many times every prefix path in $\mathcal{L}(floop)$ can be extended to an infinite path.
- Every infinite path of $floop$ visit F_M infinitely often. Hyp. FR.2 ensures that D_{n-1} underapproximates the fair states F_M. We have already shown above that every infinite path of $floop$ reaches a state D_{n_1} infinitely often. Therefore, such paths visit F_M infinitely often.
- Every step of every path in $\mathcal{L}(floop)$ satisfies T_M. Every step of every path in $\mathcal{L}(floop)$, by definition, corresponds to a transition of some funnel fnl. By hypotheses F.2, F.4, FL.1 and FL.2 every such path remains within the union of the regions and visits them following the order of the funnels. Therefore, every transition in every path of $floop$ must satisfy $S \wedge T$ for some funnel fnl in the sequence. Hyp. FR.3 ensures that if $S \wedge T$ holds that also T_M is true. Therefore every step of every path of $floop$ is also a step of M.
- $\mathcal{L}(floop)$ allows for at least one path which is a suffix of some path of M. Hyp. FR.1 ensures that there exists a finite path π_{pref} of M starting in I_M and ending in some state $\boldsymbol{v}$ such that $\boldsymbol{v} \models \bigvee_{i=0}^{n-1} S_i$. Therefore, $\boldsymbol{v}$ must be in S_j for some $0 \leq j < n$. Then, in $floop$ we can extend $\boldsymbol{v}$ to an infinite fair path π_{suf} starting in $\boldsymbol{v}$. As shown above every step of π_{suf} satisfies the transition relation of M and visits its fairness condition F_M infinitely often. The concatenation π of π_{pref} and π_{suf} without repetition of $\boldsymbol{v}$, starts

from a state in I_M, every steps satisfies T_M and visits F_M infinitely often. Therefore, π is a fair path for M: $\pi \in \mathcal{L}(M)$ and $\pi \models \mathbf{GF}F_M$.