Formal Methods in System Design (2021) 57:246–269
https://doi.org/10.1007/s10703-021-00377-1

Temporal prophecy for proving temporal properties of infinite-state systems

Oded Padon[1] · Jochen Hoenicke[2] · Kenneth L. McMillan[3] · Andreas Podelski[2] · Mooly Sagiv[4] · Sharon Shoham[4]

Accepted: 30 April 2021 / Published online: 23 July 2021

Abstract

Various verification techniques for temporal properties transform temporal verification to safety verification. For infinite-state systems, these transformations are inherently imprecise. That is, for some instances, the temporal property holds, but the resulting safety property does not. This paper introduces a mechanism for tackling this imprecision. This mechanism, which we call *temporal prophecy*, is inspired by prophecy variables. Temporal prophecy refines an infinite-state system using first-order linear temporal logic formulas, via a suitable tableau construction. For a specific liveness-to-safety transformation based on first-order logic, we show that using temporal prophecy strictly increases the precision. Furthermore, temporal prophecy leads to robustness of the proof method, which is manifested by a cut elimination theorem. We integrate our approach into the Ivy deductive verification system, and show that it can handle challenging temporal verification examples.

Part of the material of this article has appeared in a preliminary form in [37].

✉ Oded Padon
oded.padon@gmail.com

Jochen Hoenicke
hoenicke@informatik.uni-freiburg.de

Kenneth L. McMillan
kenmcm@cs.utexas.edu

Andreas Podelski
podelski@informatik.uni-freiburg.de

Mooly Sagiv
msagiv@cs.tau.ac.il

Sharon Shoham
sharon.shoham@gmail.com

1 VMware Research, Palo Alto, USA

2 University of Freiburg, Freiburg, Germany

3 University of Texas at Austin, Austin, USA

4 Tel-Aviv University, Tel Aviv-Yafo, Israel

Keywords Liveness · Prophecy variables · Firs-order temporal logic · Infinite-state systems · Deductive verification

1 Introduction

There are various techniques in the literature that transform the problem of verifying liveness of a system to the problem of verifying safety of a different system. These transformations compose the system with a device that has the known property that some safety condition σ implies liveness. The classical example of this is proving termination of a while loop with a ranking function. In this case, the device evaluates a chosen function r on loop entry, where the range of r is a well-founded set. The safety property σ is that r decreases at every iteration, which implies that the loop must terminate.

A related transformation, due to Biere et al. [6], applies to finite-state (possibly parameterized) systems. The safety property σ is, in effect, that no state occurs twice, from which we can infer termination. In the infinite-state case, this can be generalized using a function f that projects the program state onto a finite set. We can think of this as a ranking that tracks the set of unseen values of f and is ordered by set inclusion. However, the property that no value of f occurs twice is simpler to verify, since the composed device can non-deterministically guess the recurring value. In general, the effectiveness of a liveness-to-safety transformation depends strongly on the difficulty of the resulting safety proof problem.

Other methods can be seen as instances of this general approach. For example, the Terminator tool [12] might be seen as combining the ranking and the finite projection approaches. Another approach by Fang et al. applies a collection of ad-hoc devices with known liveness-to-safety properties to prove liveness of parameterized protocols [17]. Of greatest interest here, a recent paper by Padon et al. uses a dynamically chosen finite projection that depends on a finite prefix of the system's execution [36]. The approach of [31] also has some similar characteristics.

In the case of infinite-state systems, these transformations from liveness verification to safety verification are not precise reductions. That is, while safety of the transformed system implies liveness of the original system, a counterexample to the safety property does not in general imply a counterexample to liveness. For example, in the projection method, a terminating infinite-state system may have runs whose length exceeds the finite range of any chosen projection f, forcing some value to repeat.

In this paper, we show that the precision of a liveness-to-safety transformation can be usefully increased by the addition of *prophecy variables*. These variables are expressed as first-order LTL formulas. For example, suppose we augment the state of the system with a variable $r_{\Box p}$ that tracks the truth value of the proposition $\Box p$, which is true when p holds in all future states. We can soundly add two constraints to the transition system. To the transition relation, we add $r_{\Box p} \leftrightarrow (p \wedge {r_{\Box p}}')$, where ${r_{\Box p}}'$ denotes the value of the prophecy variable in the post-state. We also add the fairness constraint that $r_{\Box p} \vee \neg p$ holds infinitely often. These constraints are typical of tableau constructions that convert a temporal formula to a symbolic automaton. As we show in this paper, the additional information they provide refines the trace set of the transformed system, potentially eliminating false counterexamples.

In particular, we will show how to integrate tableau-based prophecy with the liveness-to-safety transformation of [36] that uses a history-based finite projection, referred to as *dynamic abstraction*. We show that the precision of this transformation is consequently increased. The result is that we can prove properties that otherwise would not be directly provable using the technique.

This paper makes the following contributions:

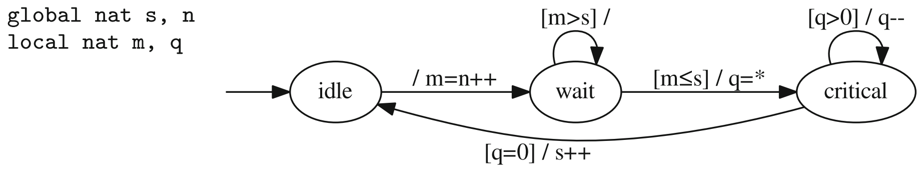
1. Introduce the notion of temporal prophecy, including prophecy formulas and prophecy witnesses, *via* a first-order LTL tableau construction.
2. Show that temporal prophecy increases the proof power (i.e., precision) of the liveness-to-safety transformation based on dynamic abstraction, and further show that the properties provable with temporal prophecy are closed under *first-order reasoning*, with cut elimination as a special case.
3. Integrate the liveness-to-safety transformation based on dynamic abstraction and temporal prophecy into the Ivy deductive verification system [34], deriving the prophecy formulas from an inductive invariant provided by the user (for proving the safety property).
4. Demonstrate the effectiveness of the approach on some challenging examples that cannot be handled by the transformation without temporal prophecy.
5. Demonstrate that prophecy witnesses can eliminate quantifier alternations in the verification conditions generated for the safety problem obtained after the transformation, facilitating decidable reasoning.

2 Illustrative example

We illustrate our approach using the ticket protocol for ensuring mutual exclusion with non starvation among multiple threads, depicted in Fig. 1. The ticket protocol may be run by any number of threads, and also allows dynamic spawning of threads. The protocol is an idealized version of spinlocks used in the Linux kernel [14]. In the protocol, each thread can be in one of three states: idle, waiting to enter the critical section, or in the critical section. The right to enter the critical section is determined by a ticket number. A global variable n, records the next available ticket, and a global variable s, records the ticket currently being served. Each thread has a local variable m that records the ticket it holds. A thread only enters the critical section when $m \leq s$. In our example, each thread has a queue of tasks to perform. Once a thread enters the critical section, it handles tasks that accumulated in its task queue, and stays in the critical section until its queue is empty (importantly, tasks are only added to the queue when the thread is outside the critical section). In Fig. 1, this is modeled by the task counter q, a thread-local variable which is non-deterministically set when a thread enters the critical section (to account for the unbounded, but finite, number of tasks), and is then decremented in each step. When $q = 0$ the thread leaves the critical section, and increments s to allow other threads to be served.

The protocol is designed to satisfy the following first-order temporal property:

$$(\forall x.\Box\Diamond scheduled(x)) \rightarrow \forall y.\Box\,(wait(y) \rightarrow \Diamond critical(y))$$



```
global nat s, n
local nat m, q
```

Fig. 1 The ticket mutual exclusion protocol. Edges are labeled by condition / action

That is, if every process is scheduled infinitely often, then every waiting process eventually enters its critical section. (Note that we encode fairness assumptions as part of the temporal property.)

Insufficiency of liveness-to-safety transformations While the temporal property is clearly satisfied by the ticket protocol, proving it is challenging for liveness-to-safety transformations. First, due to the unbounded values obtained by the ticket number and the task counter, and also due to dynamic spawning of threads, this example does not belong to the class of parameterized systems [39], where a simple lasso argument is sound (and complete) for proving liveness. Second, while using a finite abstraction can recover soundness, no fixed finite abstraction is precise enough to show the absence of a lasso-shaped counterexample in this example. The reason is that a thread can go to the waiting state (*wait*) with any number of threads waiting "ahead of it in line".

For cases where no finite abstraction is sufficiently precise to prove liveness, we may instead apply the liveness-to-safety transformation of [36]. This transformation relaxes the requirement of proving absence of lassos over a fixed finite abstraction, and instead requires one to prove absence of lassos over a *dynamic* finite abstraction that is only determined after some prefix of the trace (allowing for better precision). Soundness is maintained since the abstraction is still finite. Technically, the technique requires to prove that no *abstract lasso* exists, where an abstract lasso is a finite execution prefix that (i) visits a *freeze point*, at which a finite projection (abstraction) of the state space is fixed, (ii) the freeze point is followed by two states that are equal in the projection. We refer to these as the *repeating states*, and (iii) all fairness constraints are visited both before the freeze point and between the repeating states.

Unlike fixed finite abstractions, dynamic abstractions allow us to prove that an eventuality holds if there is a finite upper bound on the number of steps required *at the time the eventuality is asserted* (the freeze point). The bound need not be fixed *a priori*. Unfortunately, due to the non-determinism introduced by the task counter q, each of the k threads ahead of t in line could require an unbounded number of steps to leave the critical section, and this number is not yet determined when t makes its request. As a result, there is an abstract lasso which freezes the abstraction when t makes its request, after which some other thread t_0 enters the critical section and loops, decrementing its task counter q. Since the value of the task counter of t_0 is not captured in the abstraction, the loop does not change the abstract state. This spurious abstract lasso prevents this liveness-to-safety transformation from proving the property.

Temporal prophecy to the rescue The key to fixing this problem is to predict the future to the extent that a bound on the steps required for progress is determined at the freeze point. Surprisingly, this is accomplished by the use of one temporal prophecy variable corresponding to the truth value of the following formula:

$$\exists x. \Diamond\Box critical(x).$$

If this formula is initially true, there is some thread t_0 that eventually enters the critical section and stays there. At this point, we can prove it eventually exits (a contradiction) because the number of steps needed for this is bounded by the current task counter of t_0. Operationally, the freeze point is delayed until $\Box critical(x)$ holds at which point t_0's task counter is captured in the finite projection, ruling out an abstract lasso. On the other hand if the prophecy variable is initially false, then all threads are infinitely often out of the critical section. With this fairness constraint, thread t requires only a finite number of steps to be served, determined by the number of threads with lesser tickets. Operationally, the extra fairness constraint extends the lasso loop until the abstract state must change, ruling out an abstract lasso.

Though the liveness-to-safety transformation via dynamic abstraction and abstract lasso detection cannot handle the problem as given, introducing suitable temporal prophecy eliminates the spurious abstract lassos. Some spurious lassos are eliminated by postponing the freeze point, thus refining the finite abstraction, and others are eliminated by additional fairness constraints on the lasso loop. This example is explained in greater detail in Sect. 5.

3 Preliminaries

In this section, we present the first-order formalism for specifying infinite-state systems and their properties, as well as a tableau construction for first-order LTL formulas.

3.1 Transition systems in first-order logic

A first-order logic transition system is a triple (Σ, ι, τ), where Σ is a first-order vocabulary that contains only relation symbols and constant symbols (functions can be encoded by relations), ι is a closed formula over Σ defining the set of initial states, and τ is a closed formula over $\Sigma \uplus \Sigma'$, where $\Sigma' = \{\ell' \mid \ell \in \Sigma\}$, defining the transition relation. The constants in Σ represent the program variables.

A state of the transition system is a first-order structure, $s = (\mathcal{D}, \mathcal{I})$, over Σ, where $\mathcal{D}$ denotes the (possibly infinite) domain of the structure and $\mathcal{I}$ denotes the interpretation function. The set of initial states is the set of all states s such that $s \models \iota$, and the set of transitions is the set of all pairs of states (s, s') with the same domain such that $(s, s') \models \tau$. In the latter, (s, s') denotes a structure over the vocabulary $\Sigma \uplus \Sigma'$ with the same domain as s and s' in which the symbols in Σ are interpreted as in s, and the symbols in Σ' are interpreted as in s'.

For a state $s = (\mathcal{D}, \mathcal{I})$ over Σ, and for $D \subseteq \mathcal{D}$, we denote by $s|_D$ the partial structure by projecting s to D, i.e., $s|_D = (D, \mathcal{I}|_D)$, where $\mathcal{I}|_D$ interprets only constants $c \in \Sigma$ for which $\mathcal{I}(c) \in D$ (making it a partial interpretation), and for every relation symbol $r \in \Sigma$ of arity k, $\mathcal{I}|_D(r) = \mathcal{I}(r) \cap D^k$. For a vocabulary $\Sigma' \subseteq \Sigma$, we denote by $s|_{\Sigma'}$ the state over Σ' obtained by restricting the interpretation function to the symbols in Σ', i.e., $s|_{\Sigma'} = (\mathcal{D}, \mathcal{I}')$, where for every symbol $\ell \in \Sigma'$, $\mathcal{I}'(\ell) = \mathcal{I}(\ell)$.

A (finite or infinite) *trace* of (Σ, ι, τ) is a sequence of states $\pi = s_0, s_1, \ldots$ where $s_0 \models \iota$ and $(s_i, s_{i+1}) \models \tau$ for every $0 \leq i < |\pi|$. Every state along the trace has its own interpretation of the constant and relation symbols, but they all share the same domain. For a (finite or infinite) sequence of states $\pi = s_0, s_1, \ldots$, we use $\pi^i = s_i, s_{i+1}, \ldots$ for the suffix of π starting at index i.

We note that first-order transition systems are Turing-complete. Furthermore, tools such as Ivy [33,34,38] provide modeling languages that are closer to imperative programming languages and compile to a first-order transition system. This makes it easier for a user to provide a first-order specification of the transition system they wish to verify.

Safety Given a vocabulary Σ, a safety property P is a set of sequences of states over Σ, such that for every sequence of states $\pi \notin P$, there exists a finite prefix π' of π, such that π' and all of its extensions are not in P. A transition system over Σ satisfies P if all of its traces are in P.

3.2 First-order linear temporal logic (FO-LTL)

To specify temporal properties of first-order transition systems we use First-Order Linear Temporal Logic (FO-LTL), which combines LTL with first-order logic [1]. For simplicity, we consider only the "globally" ($\Box$) temporal operator. The tableau construction extends to other operators as well, and so does our approach.
Syntax Given a first-order vocabulary Σ, FO-LTL formulas are defined by:

$$
\begin{aligned}
f &::= r(t_1, \ldots, t_n) \mid t_1 = t_2 \mid \neg f \mid f_1 \vee f_2 \mid \exists x. f \mid \Box f \\
t &::= c \mid x
\end{aligned}
$$

where r is an n-ary relation symbol in Σ, c is a constant symbol in Σ, x is a variable, each t_i is a term over Σ and $\Box$ denotes the "globally" temporal operator. We also use the standard shorthand for the "eventually" temporal operator: $\Diamond f = \neg\Box\neg f$, and the usual shorthands for logical operators (e.g., $\forall x. f = \neg\exists x. \neg f$).
Semantics FO-LTL formulas over Σ are interpreted over infinite sequences of states (first-order structures) over Σ. Atomic formulas are interpreted over states, the temporal operators are interpreted as in traditional LTL, and first-order quantifiers are interpreted over the shared domain $\mathcal{D}$ of all states in the trace. Formally, the semantics is defined w.r.t. an infinite sequence of states $\pi = s_0, s_1, \ldots$ and an assignment σ that maps variables to $\mathcal{D}$ — the shared domain of all states in π. The semantics is defined as follows.

$$
\begin{aligned}
\pi, \sigma \models r(t_1, \ldots, t_n) &\Leftrightarrow s_0, \sigma \models r(t_1, \ldots, t_n) \\
\pi, \sigma \models t_1 = t_2 &\Leftrightarrow s_0, \sigma \models t_1 = t_2 \\
\pi, \sigma \models \neg\psi &\Leftrightarrow \pi, \sigma \not\models \psi \\
\pi, \sigma \models \psi_1 \vee \psi_2 &\Leftrightarrow \pi, \sigma \models \psi_1 \text{ or } \pi, \sigma \models \psi_2 \\
\pi, \sigma \models \exists x. \psi &\Leftrightarrow \text{exists } d \in \mathcal{D} \text{ s.t. } \pi, \sigma[x \mapsto d] \models \psi \\
\pi, \sigma \models \Box\psi &\Leftrightarrow \text{forall } i \geq 0, \pi^i, \sigma \models \psi
\end{aligned}
$$

When the formula has no free variables, we omit σ. A first-order transition system (Σ, ι, τ) satisfies a closed FO-LTL formula φ over Σ if all of its traces satisfy φ.

3.3 Tableau for FO-LTL

As part of our liveness-to-safety transformation, we use a standard tableau construction for FO-LTL formulas that results in a first-order transition system with *fairness constraints*. Unlike the classical construction, we define the tableau for a set of formulas, not necessarily a single temporal formula.

For an FO-LTL formula φ, we denote by $sub(\varphi)$ the set of subformulas of φ, defined in the usual way. In the sequel, we consider a finite set A of FO-LTL formulas that is closed under subformulas, i.e. for every $\varphi \in A$, $sub(\varphi) \subseteq A$. Note that A may contain formulas with free variables.

Definition 1 (Tableau vocabulary). Given a finite set A as above over a first-order vocabulary Σ, the *tableau vocabulary* for A, denoted Σ_A, is obtained from Σ by adding a fresh relation symbol $r_{\Box\varphi}$ of arity k for every formula $\Box\varphi \in A$ with k free variables.

Recall that $\Box$ is the only primitive temporal operator we consider (a similar construction can be done for other operators). The symbols added in Σ_A will be used to "label" states by temporal subformulas that are satisfied by all outgoing fair traces. To translate temporal formulas over Σ to first-order formulas over Σ_A we use the following definition.

Definition 2 For a FO-LTL formula $\varphi \in A$ (over Σ), its first-order representation, denoted $\mathrm{FO}[\varphi]$, is a first-order formula over Σ_A, defined inductively, as follows.

$$\begin{aligned}
\mathrm{FO}[\varphi] &= \varphi \text{ if } \varphi \text{ is } r(t_1,\ldots,t_n) \text{ or } t_1 = t_2\\
\mathrm{FO}[\Box\psi(\overline{x})] &= r_{\Box\psi(\overline{x})}(\overline{x})\\
\mathrm{FO}[\neg\psi] &= \neg\mathrm{FO}[\psi]\\
\mathrm{FO}[\psi_1 \vee \psi_2] &= \mathrm{FO}[\psi_1] \vee \mathrm{FO}[\psi_2]\\
\mathrm{FO}[\exists x.\psi] &= \exists x.\mathrm{FO}[\psi]
\end{aligned}$$

Note that $\mathrm{FO}[\varphi]$ has the same free variables as φ. We can now define the tableau for A as a transition system.

Definition 3 (Tableau transition system). The *tableau transition system* for A is the first-order transition system $T_A = (\Sigma_A, \mathit{true}, \tau_A)$, where τ_A (defined over $\Sigma_A \uplus \Sigma_A{}'$) is defined as follows:

$$\tau_A = \bigwedge_{\Box\varphi \in A} \forall\overline{x}.\ (r_{\Box\varphi}(\overline{x}) \leftrightarrow (\mathrm{FO}[\varphi(\overline{x})] \wedge r_{\Box\varphi}{}'(\overline{x}))).$$

Note that the original symbols in Σ (and Σ') are not constrained by τ_A, and may change arbitrarily with each transition. However, the $r_{\Box\varphi}$ relations are updated in accordance with the property that $\pi, \sigma \models \Box p$ iff $s_0, \sigma \models p$ and $\pi^1, \sigma \models \Box p$ (where s_0 is the first state of π and p is a first-order formula over Σ).

Definition 4 (Fairness). A sequence of states $\pi = s_0, s_1, \ldots$ over Σ_A is A-*fair* if for every temporal formula $\Box\varphi(\overline{x}) \in A$ and for every assignment σ, there are infinitely many i's for which $s_i, \sigma \models \mathrm{FO}[\Box\varphi(\overline{x}) \vee \neg\varphi(\overline{x})]$.

Note that $\Box\varphi(\overline{x}) \vee \neg\varphi(\overline{x})$, used above, is equivalent to $\Diamond\neg\varphi(\overline{x}) \to \neg\varphi(\overline{x})$. So the definition of fairness ensures an eventuality cannot be postponed forever. In the sequel, the set A is always clear from the context (e.g., from the vocabulary), hence we omit it and simply say that π is fair.

The next claims summarize the properties of the tableau; Lemma 1 states that the FO-LTL formulas over Σ that hold in the outgoing traces of a tableau state correspond to the first-order formulas over Σ_A that hold in the state; Lemma 2 states that every sequence of states over Σ has a representative trace in the tableau; finally, Theorem 1 states that a transition system satisfies a FO-LTL formula iff its product with the tableau of the negated formula has no fair traces.

Lemma 1 *In a fair trace $\pi = s_0, s_1, \ldots$ of T_A (over Σ_A), for every FO-LTL formula $\psi(\overline{x}) \in A$, for every assignment σ and for every index $i \in \mathbb{N}$, we have that $s_i, \sigma \models \mathrm{FO}[\psi(\overline{x})]$ iff $\pi^i, \sigma \models \psi(\overline{x})$.*

Lemma 2 *Every infinite sequence of states $\hat{s}_0, \hat{s}_1, \ldots$ over Σ can be extended to a fair trace $\pi = s_0, s_1, \ldots$ of T_A (over Σ_A) s.t. for every $i \in \mathbb{N}$, $s_i|_\Sigma = \hat{s}_i$.*

Definition 5 (Product system). Given a transition system $T_S = (\Sigma, \iota, \tau)$, a closed FO-LTL formula φ over Σ, a finite set A of FO-LTL formulas over Σ closed under subformulas such that $\neg\varphi \in A$, we define the *product system* of T_S and $\neg\varphi$ over A as the first-order transition system $T_P = (\Sigma_P, \iota_P, \tau_P)$ given by $\Sigma_P = \Sigma_A$, $\iota_P = \iota \wedge \mathrm{FO}[\neg\varphi]$ and $\tau_P = \tau \wedge \tau_A$, where $T_A = (\Sigma_A, \mathit{true}, \tau_A)$ is the tableau for A.

Theorem 1 *Let T_P be the product system of T_S and $\neg\varphi$ over A as defined in Definition 5. Then $T_S \models \varphi$ iff T_P has no fair traces.*

Intuitively, the product system augments the states of T_S with temporal formulas from A, splitting each state into many (often infinitely many) states according to the future behavior of its outgoing traces. For example, if A includes the formula $\Box r(x)$, then each state is augmented with information about the future behavior of r on every model element. If the domain is infinite, then there are infinitely many states of the product system for one state of the original system.

Note that Theorem 1 holds already when $A = sub(\neg\varphi)$. However, as we will see, taking a larger set A is useful for proving fair termination via the liveness-to-safety transformation.

4 Liveness-to-safety with temporal prophecy

In this section we present our liveness proof approach using temporal prophecy and a liveness-to-safety transformation. As in earlier approaches, our transformation (i) uses a tableau construction to construct a product transition system equipped with fairness constraints such that the latter has no fair traces iff the temporal property holds of the original system, and (ii) defines a safety property over the product transition system such that safety implies that no fair traces exist (note that the opposite direction does not hold).

The gist of our liveness-to-safety transformation is that we augment the construction of the product transition system with two forms of prophecy detailed in Sect. 4.2. We then use the definition of the safety property from [36]. In the sequel, we first present the safety property and then present the augmentation with temporal prophecy, whose goal is to "refine" the product system such that it will be safe.

4.1 Safety property: absence of abstract lassos

Given a transition system $T_W = (\Sigma_W, \iota_W, \tau_W)$ with $\Sigma_W \supseteq \Sigma_A$ (e.g., the product system from Definition 5), we define a notion of an abstract lasso, whose absence is a safety property that implies that T_W has no A-fair traces. This section recapitulates material from [36].

The definition of an abstract lasso is based on a dynamic abstraction that is fixed at some point along the trace, henceforth called the *freeze point*. The abstraction function is defined by projecting a state (a first-order structure) into a finite subset of its domain. This finite subset is defined by the union of the *footprints* of all states encountered until the freeze point, where the footprint of a state includes the interpretation it gives all constants from Σ_W. Intuitively, the footprint includes all elements "exposed" in the state, including those "touched" by outgoing transitions.

Definition 6 (Footprint). For a state $s = (\mathcal{D}, \mathcal{I})$ over Σ_W, we define the footprint of s as $f(s) = \{\mathcal{I}(c) \mid c \in \Sigma_W\}$. For a sequence of states $\pi = s_0, s_1, \ldots$ over Σ_W, and an index $i < |\pi|$, we define the footprint of $s_0, \ldots, s_i$ as $f(s_0, \ldots, s_i) = \bigcup_{j=0}^{i} f(s_j)$.

Importantly, the footprint of a finite trace always contains finitely many elements. As a result, an abstraction function that maps each state to the result of projecting it to the footprint of the trace until the freeze point has a finite range.

Definition 7 (Fair Segment). Let $\pi = s_0, s_1, \ldots$ be a sequence of states over Σ_W. For $0 \leq i \leq j < |\pi|$, we say the segment $[i, j]$ is fair if for every formula $\Box\psi(\overline{x}) \in A$, and for every assignment σ where every variable is assigned to an element of $f(s_0, \ldots, s_i)$, there exists $i \leq k \leq j$ s.t. $s_k, \sigma \models \text{FO}\,[(\Box\psi(\overline{x})) \vee \neg\psi(\overline{x})]$.

Definition 8 (Abstract Lasso). A finite trace $s_0, \ldots, s_n$ of T_W is an *abstract lasso* if there are $0 \leq i \leq j < k \leq n$ s.t. the segments $[0, i]$ and $[j, k]$ are fair, and $s_j|_{f(s_0,\ldots,s_i)} = s_k|_{f(s_0,\ldots,s_i)}$.

Intuitively, in the above definition, i is the *freeze point*, where the abstraction is fixed. The states s_j and s_k are the "repeating states" – states that are indistinguishable by the abstraction that projects them to the footprint $f(s_0, \ldots, s_i)$. The segment between j and k, respectively, the segment between 0 and i, meet all the fairness constraints restricted to elements in $f(s_0, \ldots, s_j)$, respectively, in $f(s_0)$. Fairness of the segment $[0, i]$ is needed to prevent the freeze point from being chosen too early, thus creating spurious abstract lassos. Note that the absence of abstract lassos is a safety property.

Lemma 3 *If T_W has no abstract lassos then it also has no fair traces.*

Proof Assume to the contrary that T_W has a fair trace $\pi = s_0, s_1, \ldots$. Let i be the first index such that $[0, i]$ is fair (such an index must exist since the set $f(s_0)$, which determines the relevant fairness constraints is finite). Since $f(s_0, \ldots, s_i)$ is also finite, there must exist an infinite subsequence π' of π^i such that for every s, s' in this subsequence $s|_{f(s_0,\ldots,s_i)} = s'|_{f(s_0,\ldots,s_i)}$. Let $j \geq i$ be the index in π of the first state in π'. $f(s_0, \ldots, s_j)$ is also finite, hence there exists $k' > j$ such that the segment $[j, k']$ of π is fair. Take k to be the index in π of the first state of $\pi^{k'}$ that is also in π'. Since π' is infinite, such a k must exist. Since $k \geq k'$, the segment $[j, k]$ is also fair. This defines an abstract lasso $s_0, \ldots, s_i, \ldots, s_j, \ldots, s_k$, in contradiction. □

4.2 Augmenting the transition system with temporal prophecy

In this section we explain how our liveness-to-safety transformation constructs $T_W = (\Sigma_W, \iota_W, \tau_W)$, to which we apply the safety property of Sect. 4.1. Our construction exploits both *temporal prophecy formulas* and *prophecy witnesses*, explained below. For the rest of this section we fix a first-order transition system $T_S = (\Sigma, \iota, \tau)$ and a closed FO-LTL formula φ over Σ that we wish to verify in T_S.

Temporal Prophecy Formulas First, given a set A of (not necessarily closed) FO-LTL formulas closed under subformula that contains $\neg\varphi$ (i.e., $\neg\varphi \in A$ and for any $\psi \in A$ and ψ' a subformula of ψ, $\psi' \in A$), we construct the product system $T_P = (\Sigma_P, \iota_P, \tau_P)$ defined in Definition 5. By Theorem 1, $T_S \models \varphi$ iff T_P has no fair traces. Note that classical tableau constructions are defined with $A = sub(\neg\varphi)$, and we allow A to include more formulas. These formulas act as "temporal prophecy variables" in the sense that they split the states of T_S, according to the future behavior of outgoing traces.

While the liveness-to-safety transformation is already sound with $A = sub(\neg\varphi)$, one of the chief observations of this work is that temporal prophecy formulas improve its precision. These additional formulas in A split the states of T_S into more states in T_P, and they cause some non-determinism of the future trace to be "pulled backwards" (the outgoing traces contain less non-determinism). For example, if $r_{\Box\varphi}$ holds for some elements in the current state, then φ must continue to hold for these elements in the future of the trace. Similarly, for elements where $r_{\Box\varphi}$ does not hold, there will be some time in the future of the trace where φ would not hold for them.

This is exploited by the liveness-to-safety transformation in three ways, eliminating spurious abstract lassos. First, having more temporal formulas in A refines the definition of a fair segment, and postpones the freeze point, thus making the abstraction defined by the footprint up to the freeze point more precise. For example, if $r_{\Box\varphi}$ does not hold for a ground formula φ in the initial state, then the freeze point would be postponed until after φ does not hold for the first time. Second, it strengthens the requirement on the looping segment $s_j \dots s_k$, in a similar way. Third, the additional relations in Σ_P that come from the tableau vocabulary (Σ_A) are part of the state as considered by the transformation, and a difference in these relations (projected to the footprint up to the freeze point) is a valid difference. These three ways all played a role in the examples considered in our evaluation.

Prophecy Witnesses The notion of an abstract lasso, used to define the safety property, considers a finite abstraction according to the footprint, which depends on the constants of the vocabulary. To increase the precision of the abstraction, we augment the vocabulary with fresh constants that serve as *prophecy witnesses* for existential properties.

To illustrate the idea, consider the formula $\psi(x) = \Diamond\Box p(x)$ where x is a free variable. If ψ holds for some element, it is useful to include in the vocabulary a constant that serves as a witness for $\psi(x)$, and whose interpretation will be taken into account by the abstraction. If ψ holds for some x, the interpretation of the constant will be taken from such an x. Otherwise, this constant will be allowed to take any value.

Temporal prophecy witnesses not only refine the abstraction, they can also be used in the inductive invariant. In particular, as demonstrated in Hybrid Reliable Broadcast and the TLB Shootdown examples (see Sects. 7.2.4 and 7.2.3), in some cases this allows to avoid quantifier alternation cycles in the verification conditions, leading to decidability of VC checking.

Formally, given a set $B \subseteq A$, we construct $T_W = (\Sigma_W, \iota_W, \tau_W)$ as follows. We extend Σ_P to Σ_W by adding fresh constant symbols $c_1, \dots, c_n$ for every formula $\psi(x_1, \dots, x_n) \in B$. We denote by C the set of new constants, i.e., $\Sigma_W = \Sigma_P \cup C$. The transition relation formula is extended to keep the new constants unchanged, i.e. $\tau_W = \tau_P \wedge \bigwedge_{c \in C} c = c'$, and we define ι_W by

$$\iota_W = \iota_P \wedge \mathrm{FO}\left[(\exists x_1, \dots, x_n. \psi(x_1, \dots, x_n)) \rightarrow \psi(c_1, \dots, c_n)\right]$$

Namely, $c_1, \dots, c_n$ are required to serve as witnesses for $\psi(x_1, \dots, x_n)$ in case it holds in the initial state for some elements, and otherwise they may get any interpretation at the initial state, after which their interpretation remains unchanged. Adding these fresh constants and their defining formulas to the initial state is a conservative extension, in the sense that every fair trace of T_P can be extended to a fair trace of T_W (fairness of traces over $\Sigma_W \supseteq \Sigma_A$ is defined as in Definition 4), and every fair trace of T_W can be projected to a fair trace of T_P. As such we have the following:

Lemma 4 *Let $T_P = (\Sigma_P, \iota_P, \tau_P)$ and $T_W = (\Sigma_W, \iota_W, \tau_W)$ be defined as above. Then T_P has no fair traces iff T_W has no fair traces.*

The overall soundness of the liveness-to-safety transformation is given by the following theorem.

Theorem 2 (Soundness). *Given a first-order transition system T_S and a closed FO-LTL formula φ both over Σ, and given a set of temporal prophecy formulas A over Σ that contains $\neg\varphi$ and is closed under subformula, and a set of temporal prophecy witness formulas $B \subseteq A$, if T_W as defined above does not contain an abstract lasso, then $T_S \models \varphi$.*

5 Proof of ticket with task queues

In this section we present a detailed model of the ticket protocol with task queues (introduced in Sect. 2) as a first-order transition system, show that its liveness property cannot be proved by the liveness-to-safety transformation without temporal prophecy, and show a complete proof using temporal prophecy.

5.1 Model in first-order logic

In the ticket protocol, each thread operates according to the control flow graph of Fig. 1. We model this protocol as a first-order transition system using the techniques of [33–35,38]. These techniques are implemented in the Ivy system, and benefit from decidability of the EPR fragment, allowing us to mechanically check the proof, as explained in Sect. 7. Figure 2 presents the ticket protocol model as a first-order transition system in RML syntax.

We use a vocabulary with two sorts: thread and number. The first represents threads, and the second represents ticket values and counter values. The vocabulary includes a static binary relation symbol $\leq$: number, number, with suitable first-order axioms to make it a total order, so ticket and counder values are abstracted to be a general total order. The state of the system is modeled by unary relations for the program counter: *idle*, *wait*, *critical*, constant symbols of sort number for the global variables n, s, and relations of sort thread, number for the local variables m, q. We use relations rather than functions to avoid quantifier alternation cycles. The vocabulary also includes a unary relation *scheduled*, which holds the last scheduled thread.

The model includes an action for each edge of the control flow graph of Fig. 1. The REQUEST action corresponds to an idle thread making a request to enter the critical section. The WAIT action corresponds to the self-loop of the *wait* state; this action does not change the threads' state, but it does affect the *scheduled* relation used in the temporal specification to capture the fair scheduling assumption. The ENTER action lets a thread enter the critical section if it holds a suitable ticket number. The WORK action capture the self-loop of the *critical* state, decrementing the thread's task counter. Finally, the EXIT action allows a thread with no more tasks (task counter zero) to leave the critical section, while incrementing the global variable s.

5.2 Insufficiency of liveness-to-safety without temporal prophecy

We illustrate the need for temporal prophecy by showing that without it, the liveness-to-safety transformation cannot prove liveness of the ticket protocol with task queues. Namely, we show that without temporal prophecy, an abstract lasso exists in the obtained transition system.

```
sort thread
sort number

relation $\leq$ : number, number
axiom $\forall x$ : number. $x \leq x$
axiom $\forall x$ : number, $y$ : number, $z$ : number. $x \leq y \land y \leq z \to x \leq z$
axiom $\forall x$ : number, $y$ : number. $x \leq y \land y \leq x \to x = y$
axiom $\forall x$ : number, $y$ : number. $x \leq y \lor y \leq x$

constant $zero$ : number
axiom $\forall x$ : number. $zero \leq x$

variable $n$ : number
variable $s$ : number
relation $idle$ : thread
relation $wait$ : thread
relation $critical$ : thread
relation $m$ : thread, number
relation $q$ : thread, number
relation $scheduled$ : thread

init $n = zero \land s = zero$
init $\forall t$ : thread. $idle(t) \land \neg wait(t) \land \neg critical(t) \land \neg scheduled(t)$
init $\forall t$ : thread, $k$ : number. $(m(t,k) \leftrightarrow k = zero) \land (q(t,k) \leftrightarrow k = zero)$

action REQUEST($t$ : thread) {
  assume $idle(t)$
  $m(t,X) := X = n$
  $n := succ(n)$
  $idle(t) :=$ false
  $wait(t) :=$ true
  $scheduled(T) := T = t$
}
action WAIT($t$ : thread, $k$ : number) {
  assume $wait(t) \land m(t,k) \land k > s$
  $scheduled(T) := T = t$
}
action ENTER($t$ : thread, $k$ : number, $j$ : number) {
  assume $wait(t) \land m(t,k) \land k \leq s$
  $q(t,X) := X = j$
  $wait(t) :=$ false
  $critical(t) :=$ true
  $scheduled(T) := T = t$
}
action WORK($t$ : thread, $j$ : number) {
  assume $critical(t) \land q(t, succ(j))$
  $q(t,X) := X = j$
  $scheduled(T) := T = t$
}
action EXIT($t$ : thread) {
  assume $critical(t) \land q(t, zero)$
  $s := succ(s)$
  $critical(t) :=$ false
  $idle(t) :=$ true
  $scheduled(T) := T = t$
}

specification $(\forall x$ : thread. $\Box\Diamond scheduled(x)) \to \forall y$ : thread. $\Box\,(wait(y) \to \Diamond critical(y))$
```

Fig. 2 RML model of the ticket protocol with task queues. $succ(a)$ is a macro used to obtain the successor of a ground term a by introducing a fresh logical constant b and assuming the formula $a < b \land \forall x.\ a < x \to b \leq x$. The five actions match the five edges of the control flow graph depicted in Fig. 1

Suppose we do not augment the tableau with additional prophecy. That is, take A to contain just the subformulas of the temporal property:

$$(\forall x.\Box\Diamond scheduled(x)) \to \forall y.\Box\,(wait(y) \to \Diamond critical(y))$$

We observe that the liveness-to-safety transformation results in a system T_W that contains an abstract lasso. This is regardless of the choice of B. To see this, consider the following trace with two threads denoted t_1 and t_2:[1]

1. Thread t_1 enters the wait state, obtaining ticket 0 (sets $m(t_1, 0)$) and increasing n to 1;
2. Thread t_2 enters the wait state, obtaining ticket 1 (sets $m(t_2, 1)$) and increasing n to 2;
3. Thread t_1 enters the critical section, setting its task counter to 4 (sets $q(t_1, 4)$);
4. Thread t_2 is scheduled, and stays in the wait state;
5. Thread t_1 is scheduled, decreasing its counter (sets $q(t_1, 3)$) and staying in the critical section.

This sequence forms an abstract lasso, with the freeze point occuring after step 2, and the states before step 4 and after step 5 being the same under the projection. The freeze point occurs after step 2, before t_1 enters the critical section (in this point both threads have already been scheduled, making the segment fair). Thus, the footprint contains only thread values $\{t_1, t_2\}$ and number values $\{0, 1, 2\}$. The projection of q to the footprint turns effectively turns this relation from a total function to a partial function that is undefined for a thread whose counter value is larger than 2. Because of this, steps 4 and 5 form the loop of the abstract lasso. The segment of steps 4 and 5 satisfies the fairness constraints (both threads are scheduled), and the starting and ending states agree on the value of all relations for elements in the footprint. The only difference between the state prior to step 4 and the state after step 5 is that the task counter of t_1 changed from 4 to 3, but this change is invisible under the projection to the footprint $\{t_1, t_2, 0, 1, 2\}$.

5.3 Proof with temporal prophecy

Next we show that when adding the temporal prophecy formula $\exists x. \Diamond\Box critical(x)$ to the tableau construction, no abstract lasso exists in the augmented transition system, hence the liveness-to-safety transformation succeeds to prove the property.

With the formula $\exists x. \Diamond\Box critical(x)$ added as temporal prophecy, A includes the following two formulas and their subformulas:

$$\neg\left((\exists x. \neg\Box\neg\Box\neg scheduled(x)) \vee \neg\exists x. \neg\Box\left(\neg wait(x) \vee \neg\Box\neg critical(x)\right)\right)$$
$$\exists x. \neg\Box\neg\Box critical(x)$$

And $B = \{\neg\Box\left(\neg wait(x) \vee \neg\Box\neg critical(x)\right)\ ,\ \neg\Box\neg\Box critical(x)\}$. When reading these formulas, it is useful to keep the following identities in mind:

$$\Box\neg\Box\neg scheduled(x) \equiv \Box\Diamond scheduled(x)$$
$$\Box\left(\neg wait(x) \vee \neg\Box\neg critical(x)\right) \equiv \Box\left(wait(y) \to \Diamond critical(y)\right)$$
$$\neg\Box\neg\Box critical(x) \equiv \Diamond\Box critical(x)$$

With A and B as above, Σ_W extends the original vocabulary with the following 6 unary relations:

$$r_{\Box\neg scheduled(x)},\ r_{\Box\neg\Box\neg scheduled(x)},\ r_{\Box\neg critical(x)}, r_{\Box\neg wait(x)\vee\neg\Box\neg critical(x)},$$
$$r_{\Box critical(x)},\ r_{\Box\neg\Box critical(x)};$$

[1] We use 0, 1, 2, 3, 4 to denote elements of the number sort, with the intention that the $\leq$ relation is interpreted as expected.

as well as two constant symbols for temporal prophecy witnesses: c_1 for the formula $\neg\Box\,(\neg \mathit{wait}(x) \vee \neg\Box\neg \mathit{critical}(x))$, and c_2 for the formula $\neg\Box\neg\Box \mathit{critical}(x)$.

We now explain why there is no abstract lasso. To do this, we show that the tableau construction, combined with the dynamic abstraction and the fair segment requirements, result in the same reasoning that was presented informally in Sect. 2.

First, observe that from the definition of c_1 and the negation of the liveness property (both assumed by ι_W), we have that the initial state $s_0 \models \mathrm{FO}\,[\neg\Box\,(\neg \mathit{wait}(c_1) \vee \neg\Box\neg \mathit{critical}(c_1))]$. For brevity, denote $p = (\neg \mathit{wait}(c_1) \vee \neg\Box\neg \mathit{critical}(c_1))$, so we have $s_0 \models \mathrm{FO}\,[\neg\Box p]$, i.e., $s_0 \models \neg r_{\Box p}$. Since c_1 is also in the footprint of the initial state, the fair segment requirement ensures that the freeze point can only happen after encountering a state satisfying: $\mathrm{FO}\,[(\Box p) \vee \neg p] \equiv r_{\Box p} \vee \mathrm{FO}\,[\neg p]$. Recall that the transition relation of the tableau (τ_A), ensures $(r_{\Box p}) \leftrightarrow (\mathrm{FO}\,[p] \wedge {r_{\Box p}}')$. Therefore, on update from a state satisfying $\neg r_{\Box p}$ to a state satisfying $r_{\Box p}$ can only happen if the pre-state satisfies $\mathrm{FO}\,[\neg p]$. Therefore, the freeze point must come after encountering a state that satisfies $\mathrm{FO}\,[\neg p] \equiv \mathit{wait}(c_1) \wedge r_{\Box\neg \mathit{critical}(c_1)}$. From the freeze point onward, τ_A will ensure both $r_{\Box\neg \mathit{critical}(c_1)}$ and $\neg \mathit{critical}(c_1)$ continue to hold, so c_1 will stay in *wait* (since the protocol does not allow to go from *wait* to anything but *critical*). So, we see that the mechanism of the tableau, combined with the temporal prophecy witness and the fair segment requirement, ensures that the freeze point happens after c_1 makes a request that is never granted. This will ensure that the footprint used for the dynamic abstraction will include all threads ahead of c_1 in line, i.e., those with smaller ticket numbers.

As for c_2, the initial state will either satisfy $\mathrm{FO}\,[\neg\Box\neg\Box \mathit{critical}(c_2)]$ or it would satisfy $\mathrm{FO}\,[\neg\exists x.\neg\Box\neg\Box \mathit{critical}(x)]$. In the first case, by an argument similar to the one used above for c_1, the freeze point will happen after c_2 enters the critical section and then stays in it. Therefore, the footprint used for the dynamic abstraction will include all numbers smaller than q of c_2 when it enters the critical section[2]. Since c_2 is required to be scheduled between the repeating states (again by the tableau construction and the fair segment requirement), its value for q will be decreased, and this will be visible in the dynamic abstraction. Thus, in this case, an abstract lasso is not possible.

In the second case the initial state satisfies $\mathrm{FO}\,[\neg\exists x.\neg\Box\neg\Box \mathit{critical}(x)]$. By a similar argument that combines the tableau with the fair segment requirement for the repeating states, we will obtain that between the repeating states, any thread in the footprint of the first repeating state, must both be scheduled and visit a state outside the critical section. In particular, this includes all threads that are ahead of c_1 in line. This entails a change to the program counter of one of them (the one that had a ticket number equal to the service number at the first repeating state), which will be visible in the abstraction. Thus, an abstract lasso is not possible in this case either.

6 Closure under first-order reasoning

The transformation from temporal verification to safety verification developed in Sect. 4 introduces an abstraction, and incurs a loss of precision. That is, for some systems and properties, liveness holds but the safety of the resulting system does not hold, no matter what temporal prophecy is used. (This is unavoidable for a transformation from arbitrary FO-LTL properties to safety properties [36].) However, in this section, we show that the set

[2] When modeling natural numbers in first-order logic, the footprint is adjusted to include all numbers lower than any constant (still being a finite set).

of instances for which the transformation can be made precise (via temporal prophecy) is closed under first-order reasoning. This is unlike the transformation of [36]. It shows that the use of temporal prophecy results in a particular kind of robustness.

We consider a proof system in which the above transformation is performed and the resulting safety property is checked by an oracle. That is, for a transition system T_S and a temporal property φ (a closed FO-LTL formula), we write $T_S \vdash \varphi$ if there exist finite sets of FO-LTL formulas A and B satisfying the conditions of Theorem 2, such that resulting transition system T_W is safe, i.e., does not contain an abstract lasso. We now show that the relation $\vdash$ satisfies a powerful closure property.

Theorem 3 (Closure under first-order reasoning). *Let T_S be a transition system, and $\psi, \varphi_1, \ldots, \varphi_n$ be closed FO-LTL formulas, such that* $\mathrm{FO}[\varphi_1 \wedge \ldots \wedge \varphi_n] \models \mathrm{FO}[\psi]$. *If $T_S \vdash \varphi_i$ for all $1 \leq i \leq n$, then $T_S \vdash \psi$.*

The condition that $\mathrm{FO}[\varphi_1 \wedge \ldots \wedge \varphi_n] \models \mathrm{FO}[\psi]$ means that $\varphi_1 \wedge \ldots \wedge \varphi_n$ entails ψ when using only first-order reasoning, and treating temporal operators as uninterpreted. The theorem states that provability using the liveness-to-safety transformation is closed under such reasoning. Two special cases of Theorem 3 given by the following corollaries:

Corollary 1 (Modus Ponens). *If T_S is a transition system and φ and ψ are closed FO-LTL formulas such that $T_S \vdash \varphi$ and $T_S \vdash \varphi \rightarrow \psi$, then $T_S \vdash \psi$.*

Corollary 2 (Cut). *If T_S is a transition system and φ and ψ are closed FO-LTL formulas such that $T_S \vdash \varphi \rightarrow \psi$ and $T_S \vdash \neg\varphi \rightarrow \psi$, then $T_S \vdash \psi$.*

Proof of Theorem 3 In the proof we use the notation $T_W(T_S, \varphi, A, B)$ to denote the transition system constructed for T_S and φ when using A, B as temporal prophecy formulas. Likewise, we refer to the vocabulary, initial states and transition relation formulas of the transition system as $\Sigma_W(T_S, \varphi, A, B)$, $\iota_W(T_S, \varphi, A, B)$, and $\tau_W(T_S, \varphi, A, B)$, respectively. Let $(A_1, B_1), \ldots, (A_n, B_n)$ be such that $T_W(T_S, \varphi_i, A_i, B_i)$ has no abstract lasso, for every $1 \leq i \leq n$. Now, let $A = \bigcup_{i=1}^{n} A_i$ and $B = \bigcup_{i=1}^{n} B_i$. We show that $T_W(T_S, \psi, A, B)$ has no abstract lasso. Assume to the contrary that $s_0, \ldots, s_i, \ldots, s_j, \ldots, s_k, \ldots, s_n$ is an abstract lasso for $T_W(T_S, \psi, A, B)$. Since $s_0 \models \iota_W(T_S, \psi, A, B)$, we know that $s_0 \models \neg\mathrm{FO}[\psi]$, and since $\mathrm{FO}[\varphi_1 \wedge \ldots \wedge \varphi_n] \models \mathrm{FO}[\psi]$, there must be some $1 \leq \ell \leq n$ s.t. $s_0 \models \neg\mathrm{FO}[\varphi_\ell]$. Denote $\Sigma' = \Sigma_W(T_S, \varphi_\ell, A_\ell, B_\ell)$. Now, $s_0|_{\Sigma'}, \ldots, s_i|_{\Sigma'}, \ldots, s_j|_{\Sigma'}, \ldots, s_k|_{\Sigma'}, \ldots, s_n|_{\Sigma'}$ is an abstract lasso of $T_W(T_S, \varphi_\ell, A_\ell, B_\ell)$, which is a contradiction. To see that, we first simplify the notation and denote $s_m|_{\Sigma'}$ by $\hat{s}_m$. The footprint $f(s_0, \ldots, s_i)$ contains more elements than the footprint $f(\hat{s}_0, \ldots, \hat{s}_i)$, since $\Sigma_W(T_S, \psi, A, B) \supseteq \Sigma_W(T_S, \varphi_\ell, A_\ell, B_\ell)$. Therefore, given that $s_j|_{f(s_0,\ldots,s_i)} = s_k|_{f(s_0,\ldots,s_i)}$, we have that $\hat{s}_j|_{f(\hat{s}_0,\ldots,\hat{s}_i)} = \hat{s}_k|_{f(\hat{s}_0,\ldots,\hat{s}_i)}$ as well. Moreover, the fairness constraints in $T_W(T_S, \varphi_\ell, A_\ell, B_\ell)$, determined by A_ℓ, are a subset of those in $T_W(T_S, \psi, A, B)$), determined by A, so the segments $[0, i]$ and $[j, k]$ are also fair in $T_W(T_S, \varphi_\ell, A_\ell, B_\ell)$. □

The proof of Theorem 3 sheds more light on the power of using temporal prophecy formulas that are not subformulas of the temporal property to prove. In particular, the theorem does not hold if A is restricted to subformulas of the temporal proof goal.

7 Implementation and evaluation

7.1 Implementation in Ivy

We have implemented our approach for temporal verification and integrated it into the Ivy deductive verification system [34]. This allows the user to model the transition system in the Ivy language (which internally translates into a first-order transition system), and express temporal properties directly in FO-LTL. In our implementation, the safety property that results from the liveness-to-safety transformation is proven by a suitable inductive invariant, provided by the user. To facilitate this process, Ivy internally constructs a suitable monitor for the safety property, i.e., the absence of abstract lasso's in T_W. The user then provides an inductive invariant for T_W composed with this monitor. The monitor keeps track of the footprint and the fairness constraints, and non-deterministically selects the freeze point and repeated states of the abstract lasso. Similar to the construction of [6], the monitor keeps a shadow copy of the "saved state", which is the first of the two repeated states. These are maintained via designated relation symbols (in addition to Σ_W). The user's inductive invariant must then prove that it is impossible for the monitor to detect an abstract lasso.

Mining Temporal Prophecy from the Invariant As presented in previous sections, our liveness-to-safety transformation is parameterized by sets of formulas A and B. In the implementation, these sets are implicit, and are extracted automatically from the inductive invariant provided by the user. Namely, the inductive invariant provided by the user contains temporal formulas, and also prophecy witness constants, where every temporal formula $\Box\varphi$ is a shorthand (and is internally rewritten to) $r_{\Box\varphi}$. The set A to be used in the construction is defined by all the temporal subformulas that appear in the inductive invariant (and all their subformulas), and the set B is defined according to the prophecy witness constants that are used in the inductive invariant.

In particular, the user's invariant may refer to the satisfaction of each fairness constraint FO $[\Box\varphi \vee \neg\varphi]$ for $\Box\varphi \in A$, both before the freeze point and between the repeated states, via a convenient syntax provided by Ivy.

Interacting with Ivy If the user provides an inductive invariant that is not inductive, Ivy presents a graphical counterexample to induction. This guides the user to adjust the inductive invariant, which may also lead to new formulas being added to A or B, if the user adds new temporal formulas or prophecy witnesses to the inductive invariant. In this process, the user's mental image is of a liveness-to-safety transformation where A and B include all (countably many) FO-LTL formulas over the system's vocabulary, so the user is free to use any temporal formula, or prophecy witness for any formula. However, since the user's inductive invariant is a finite formula, the liveness-to-safety transformation needs only to be applied to finite A and B, and the infinite A and B are just a mental model.

7.2 Verified protocols

We have used our implementation to prove liveness for several challenging examples, summarized in Fig. 3. Our examples are publicly available[3]. We focused on examples that were beyond reach for the liveness-to-safety transformation of [36]. In [36], such examples were handled using a nesting structure. Our experience shows that with temporal prophecy, the

[3] https://github.com/kenmcmil/ivy

invariants are simpler than with a nesting structure (for additional comparison with nesting structure see Sect. 8). For all examples we considered, the verification conditions are in a decidable fragment of first-order logic which is supported by Z3 (the stratified extension of EPR [20,38]). Interestingly, for the TLB shootdown example, the proof presented in [36] (using a nesting structure) required non-stratified quantifier alternation, which is eliminated by the use of temporal prophecy witnesses. Due to the decidability of verification conditions, Z3 behaves predictably, and whenever the invariant is not inductive it produces a finite counterexample to induction, which Ivy presents graphically. Our experience shows that the graphical counterexamples provide valuable guidance towards finding an inductive invariant, and also for coming up with temporal prophecy formulas as needed. Below we provide more detail on each example.

7.2.1 Ticket with task queues

The ticket example has been discussed in Sects. 1, and 5 contains more details about its proof with temporal prophecy, using a single temporal prophecy formula and two prophecy witness constants. To give a flavor of what the proof looks like in Ivy, we present a couple of the conjectures that make up the inductive invariant for the resulting system, in Ivy's syntax. In Ivy, the prefix `l2s` indicates symbols that are introduced by the liveness-to-safety transformation.

Some conjectures are needed to state that the footprint used in the dynamic abstraction contains enough elements. An example of such a conjecture is:

```
l2s_frozen & (globally critical(c2)) -> forall N. N <= q(c2) -> l2s_a(N)
```

This conjecture states that after the freeze point (indicated by the special symbol `l2s_frozen`), if the prophecy witness `c2` (which is the prophecy witness defined for $\Diamond\Box \mathit{critical}(x)$) is globally in the critical section, then the finite domain of the frozen abstraction (stored in the unary relation `l2s_a`) contains all numbers up the `c2`'s value for `q`. Other conjectures are needed to show that the current state is different from the saved state. One example is:

```
l2s_saved & (globally critical(c2)) & ~($l2s_w X. scheduled(X))(c2) ->
        q(c2) ~= ($l2s_s X. q(X))(c2)
```

The special operator `$l2s_w` lets the user query whether a fairness constraint has been encountered, and `$l2s_s` exposes to the user the saved state. Both `$l2s_w` and `$l2s_s` are syntactically λ-like binders, so the variable `X` (of sort `thread`, which is inferred) is always bound in the above conjecture. This conjecture states that after we saved a shadow state (indicated by `l2s_saved`), if the prophecy witness `c2` is globally in the critical section, and if we have encountered the fairness constraints associated with $\mathit{scheduled}(x) \vee \Box\neg \mathit{scheduled}(x)$ instantiated for `c2` (which can only happen after `c2` has been scheduled), then the current value `c2` has for `q` is different from the same value in the shadow state.

7.2.2 Alternating bit protocol

The alternating bit protocol is a classic communication algorithm for transition of messages using lossy first-in-first-out (FIFO) channels. The protocol uses two channels: a data channel from the sender to the receiver, and an acknowledgment channel from the receiver to the sender. The sender and the receiver each have a state bit, and messages include a bit that functions as a "sequence number". We assume that the sender has an (infinite) array of

Protocol	**# A**	**# B**	**# LOC**	**# C**	**FO-LTL**	**t** [sec]
Ticket With Task Queues	1	2	90	60	22%	7
Alternating Bit Protocol	4	1	143	70	40%	25
Hybrid Reliable Broadcast	0	3	166	58	20%	25
TLB Shootdown	6	3	468	102	49%	240

Fig. 3 Protocols for which we verified liveness. For each protocol, **# A** reports the number of temporal prophecy formulas used. **# B** reports the number of prophecy witnesses used. **# LOC** reports the number of lines of code for the system model (without proof) in Ivy's modeling language. **# C** reports the number of conjectures used in the inductive invariant (a typical conjecture is one or few lines). **FO-LTL** reports the fraction of the conjectures that use temporal formulas. Finally, **t** reports the run time (in seconds) for checking the verification conditions using Ivy and Z3. The experiments were performed on a laptop with an Intel Core i7-8650U CPU, using Z3 version 4.8.4

values to send, which is filled by some independent process. The liveness property we wish to prove is that every value entered into the sender array is eventually received by the receiver.

The protocol is live under fair scheduling assumptions, as well as standard fairness constraints for the channels: if messages are infinitely often sent, then messages are infinitely often received. This makes the structure of the temporal property more involved. Formally, the liveness property we prove is:

$$\begin{aligned}&(\Box\Diamond \mathit{sender\_scheduled}) \wedge (\Box\Diamond \mathit{receiver\_scheduled}) \wedge \\ &\quad((\Box\Diamond \mathit{data\_sent}) \to (\Box\Diamond \mathit{data\_received})) \wedge \\ &\quad((\Box\Diamond \mathit{ack\_sent}) \to (\Box\Diamond \mathit{ack\_received})) \to \\ &\quad\forall x. \Box(\mathit{sender\_array}(x) \neq \bot \to \Diamond \mathit{receiver\_array}(x) \neq \bot))\end{aligned}$$

This property cannot be proven without temporal prophecy, due to a spurious abstract lasso [36, Section 5.2]. However, it can be proven using 4 temporal prophecy formulas that eliminate the spurious lassos:

$$\{\Diamond\Box\,(\mathit{sender\_bit} = s \wedge \mathit{receiver\_bit} = r) \mid s, r \in \{0, 1\}\}\,.$$

Intuitively, these formulas make a distinction between traces in which the sender and receiver bits eventually become fixed, and traces in which they change infinitely often.

7.2.3 Hybrid reliable broadcast

Hybrid Reliable Broadcast is a distributed reliable broadcast [42] protocol that is designed to tolerates four different types of faults: Byzantine faults, symmetric faults, clean crash, and crash faults. This protocol constitutes the core of the clock synchronization algorithm presented in [48]. The protocol relies on set cardiality thresholds, which we model in the EPR fragment of first-order logic using the technique of [5].

As formalized in [29], the protocol's correctness specification is composed of three properties: (i) *Unforgeability*: if no correct process receives the external message, then no correct process ever accepts; (ii) *Correctness*: if all correct processes receive the external message, then some correct process eventually accepts; and (iii) *Relay*: whenever a correct process accepts, all correct processes will eventually accept. While Unforgeability is a safety property, both Correctness and Relay are liveness properties. While these liveness proofs do not require additional temporal prophecy formulas, they do require temporal prophecy witnesses, as explained below.

Our first attempt to prove Correctness and Relay resulted in inductive invariants with quantifier alternation cycles, but we were able to use temporal prophecy witnesses to simplify the inductive invariants and eliminate the cycles, such that ultimately all verification conditions are in a decidable fragment. This required the use of one prophecy witness for Correctness, and two witnesses for Relay. Coming up with these extra prophecy witnesses and using them to simplify the invariants is a manual process, but it is guaranteed to be sound, since the resulting invariants are checked by Ivy.

To illustrate the process of using prophecy witnesses to eliminate quantifier alternation cycles, consider the following property required for the inductive invariant: if a correct process received at least $t_B + t_S + 1$ internal messages, then it sent an internal message (where t_B and t_S are bounds on the number of Byzantine-faulty and symmetric-faulty processes respectively). This means that for any correct process p that did not send an internal message and any set S of at least $t_B + t_S + 1$ processes, there must exist some process $q \in S$ such that p did not receive a message from q. This property translates (using the encoding of [5]) to:

$$\forall p : \mathsf{process}.\ \forall S : \mathsf{quorum}.\ \exists q : \mathsf{process}.\ \varphi(p, S, q),$$

where $\varphi(p, S, q)$ is quantifier-free. This results in a quantifier alternation cycles (self-loop), since we have $\forall\mathsf{process}\exists\mathsf{process}$. The cycle can be eliminated by introducing a temporal prophecy witness, based on the observation that the proof relies on the above property to show that all correct processes must eventually send an internal message. Thus, by defining w to be a witness for the formula:

$$\psi(x : \mathsf{process}) = \mathit{correct}(x) \wedge \Box\neg \mathit{sent}(x),$$

we can replace the cyclic invariant above by:

$$\forall S : \mathsf{quorum}.\ \exists q : \mathsf{process}.\ \varphi(w, S, q),$$

which does not lead to quantifier-alternation cycles.

7.2.4 TLB shootdown

The TLB shootdown algorithm [7] is used (e.g. in the Mach operating system) to maintain consistency of Translation Look-aside Buffers (TLB) across processors. When some processor (dubbed the initiator) changes the page table, it interrupts all other processors currently using the page table (dubbed the responders) and waits for them to receive the interrupt before making changes. The liveness property we prove is that no processor can become stuck either as an initiator or as a responder (formally, it will respond or initiate infinitely often). This liveness depends on fair scheduling assumptions, as well as strong fairness assumptions for the page table locks used by the protocol.

The algorithm itself is described by a state machine with 24 locations and 27 transitions, closely following the presentation in [26]. To prove liveness we assume that there is a process that gets stuck and show that there is no abstract lasso. We use one witness for the process that does not satisfy the liveness property. Since this process may lock a pagemap, we need to show that the lock operation will eventually succeed. Thus, we use a second witness for a pagemap that is never unlocked, if this exists. A third witness is used for a process that possibly gets stuck in the critical section protected by the pagemap lock. This can either be the first process or a second process blocking the first process by holding the pagemap lock. These witnesses allow for expressing all invariants required to prove the safety property in the EPR fragment. As an example, instead of requiring that for every locked pagemap there is a process in the critical region holding the lock, we only require this for the pagemap that is captured by the second witness.

We use several prophecy formulas to case split on where some process may get stuck. For example, one prophecy formula $\Diamond\Box pc(c_1) = i_2$ is used to indicate that the first process gets stuck while trying to take the lock transition from location i_2. Another prophecy formula $\Diamond\Box pc(c_3) \in \{i_3, \ldots, i_8\}$ expresses that the process c_3 gets stuck in the loop of the algorithm spanning the locations $i_3, \ldots, i_8$. For the pagemap c_2 there is a prophecy formula $\Diamond\Box plock(c_2)$ to indicate that it is locked forever.

We then show that if the first process gets stuck, one of the prophecy variables that indicate where this process gets stuck must hold. If the first process is not stuck in any loop or in the lock operation, its program counter, which is part of the abstraction, indicates progress each time the first process is scheduled. If the first process gets stuck while taking the lock, strong fairness implies that the prophecy variable indicating that a page is locked forever holds. This in turn implies that the second process, which locks the page must get stuck and one of the corresponding prophecy variables holds. If the prophecy variable holds that indicate that either the first or the second process gets stuck in one of the loops, we show the absence of abstract lassos. The loops iterate over all active processes to send them an interrupt or to check that they received the interrupt. To prove their termination, we utilize the fact that the finite abstraction contains all processes the loop iterates over, because it is fixed at the time the stuck process has entered the loop. After each loop iteration, the corresponding process is marked as processed, which changes the abstraction. For the instruction that waits for another process to receive the interrupt, we note that while that other process has not yet indicated reception of the interrupt it moves only forward and each time it is scheduled the abstraction changes.

Compared to the proof of [36], our proof is simpler due to the temporal prophecy, and avoids non-stratified quantifier alternation, resulting in verification conditions that are in a decidable fragment.

8 Related work

Prophecy variables were first introduced in [2], in the context of refinement mappings. There, prophecy variables are required to range over a finite domain to ensure soundness. Our notion of prophecy via first-order temporal formulas and witness constants does not meet this criterion, but is still sound as assured by Theorem 2. In [27], LTL formulas are used to define prophecy variables in a way that is similar to ours, but only to show refinement between finite-state processes. We use temporal prophecy defined by FO-LTL formulas in the context of infinite-state systems. Furthermore, we consider a liveness-to-safety transformation (rather than refinement mappings), which can be seen as a proof system for FO-LTL.

The liveness-to-safety transformation based on dynamic abstraction, but *without temporal prophecy*, was introduced in [36]. There, a *nesting structure* was used to increase the power of the transformation. A nesting structure is defined by the user (via first-order formulas), and has the effect of splitting the transition system into levels (analogous to nested loops) and proving each level separately. Temporal prophecy as we introduce here is more general, and in particular, any proof that is possible with a nesting structure, is also possible with temporal prophecy (by adding a temporal prophecy formula $\Diamond\Box\delta$ for every nesting level, defined by δ). Moreover, the nesting structure does not admit cut elimination or closure under first-order reasoning, and is therefore less robust.

One effect of prophecy is to split cases in the proof on some aspect of the future. This very general idea occurs in various approaches to liveness, particularly in the large body of work on lexicographic or disjunctive rankings for termination [4,8,11–13,15,19,21,22,24,28,30,40, 41,43–46]. In the work of [23], the partitioning of the space of potentially infinite executions is based on the *a priori* decomposition of regular expressions for iterated loop segments. Often the partitioning here amounts to a split according to a fairness condition ("command *a* is taken infinitely often or it is not"). The partitioning is constructed dynamically (and represented explicitly through a union of Buchi automata) in [25] (for termination), in [16] (for liveness), and in [18] (for liveness of parameterized systems). None of these works uses a temporal tableau construction to partition the space of futures, however.

Here, we use prophecy to, in effect, partially determinize a system by making non-deterministic choices earlier in an execution. This same effect was used for a different purpose in refining an abstraction from LTL to ACTL [10] and checking CTL* properties [9]. The prophecy in this case relates only to the next transition and is not expressed temporally. The method of "temporal case splitting" in [32] can also be seen as a way to introduce prophecy variables to increase the precision of an abstraction, though in that case the transformation was to finite-state liveness, not infinite-state safety. Moreover, it only introduces temporal witnesses.

We have considered only proof methods that transform liveness to safety (which includes the classical ranking approach for while loops). There are approaches, however, which do *not* transform liveness to safety. For example, the approaches in [3,15,47] are essentially forms of widening in a CTL-style backwards fixpoint iteration. It is not clear to what extent temporal prophecy might be useful in increasing the precision of such abstractions, but it may be an interesting topic for future research.

9 Conclusion

We have seen that the addition of prophecy variables in the form of temporal formulas can increase the precision of liveness-to-safety tranformations for infinite-state systems. The prophecy variables are derived from additional temporal formulas that in our implementation were mined from the invariants a user provides to prove the safety property. This approach is effective for proving challenging examples. By increasing the precision of the dynamic abstraction, it avoided the need to decompose the proof into nested termination arguments, reducing the human effort of proof construction. Though completeness is not possible, we saw that the additional expressiveness of temporal prophecy provides a cut elimination property. While we considered temporal prophecy using a particular liveness-to-safety construction (based on dynamic abstraction), it seems reasonable to expect that the tableau-based approach would apply to other constructions and abstractions, including constructions based on rankings and well-founded relations. Because our approach relies on an inductive invariant supplied by the user, it requires the user to understand the liveness-to-safety transformation and it requires both cleverness and a deep understanding of the protocol. For this reason, a possible avenue for future research would be to explore invariant synthesis techniques, and in particular ones that account for refinement due to temporal prophecy.

Acknowledgements We thank the anonymous referees for insightful comments which improved this paper. Padon was supported by Google under a PhD fellowship. Padon and Sagiv were supported by the European Research Council under the European Union's Seventh Framework Program (FP7/2007–2013) / ERC grant agreement no. [321174-VSSC]. This publication is part of a project that has received funding from the European

Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No [759102-SVIS]). The research was partially supported by Len Blavatnik and the Blavatnik Family foundation, and by the Blavatnik Interdisciplinary Cyber Research Center, Tel Aviv University. This material is based upon work supported by the United States-Israel Binational Science Foundation (BSF) grants No. 2016260 and 2012259.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Abadi M (1989) The power of temporal proofs. Theor Comput Sci 65(1):35–83. https://doi.org/10.1016/0304-3975(89)90138-2
2. Abadi M, Lamport L (1991) The existence of refinement mappings. Theor Comput Sci 82(2):253–284. https://doi.org/10.1016/0304-3975(91)90224-P
3. Abdulla PA, Jonsson B, Rezine A, Saksena M (2006) Proving liveness by backwards reachability. In: CONCUR, lecture notes in computer science, vol 4137. Springer, pp 95–109
4. Babic D, Hu AJ, Rakamaric Z, Cook B (2007) Proving termination by divergence. In: SEFM, pp 93–102
5. Berkovits I, Lazic M, Lossa G, Padon O, Shoham S (2019) Verification of threshold-based distributed algorithms by decomposition to decidable logics. In: Computer aided verification—31th international conference, CAV
6. Biere A, Artho C, Schuppan V (2002) Liveness checking as safety checking. Electr Notes Theor Comput Sci 66(2):160–177
7. Black DL, Rashid RF, Golub DB, Hill CR (1989) Translation lookaside buffer consistency: a software approach. In: Proceedings of the third international conference on architectural support for programming languages and operating systems, ASPLOS III. ACM, New York, NY, USA, pp 113–122. https://doi.org/10.1145/70082.68193
8. Brockschmidt M, Cook B, Fuhs C (2013) Better termination proving through cooperation. In: Proceedings of the computer aided verification—25th international conference, CAV 2013, Saint Petersburg, Russia, July 13–19, 2013, pp 413–429
9. Cook B, Khlaaf H, Piterman N (2015) On automation of ctl* verification for infinite-state systems. In: Kroening D, Pasareanu CS (eds) Proceedings of the computer aided verification—27th international conference, CAV 2015, San Francisco, CA, USA, July 18–24, 2015, Part I, lecture notes in computer science, vol 9206. Springer, pp 13–29. https://doi.org/10.1007/978-3-319-21690-4_2
10. Cook B, Koskinen E (2011) Making prophecies with decision predicates. In: Ball T, Sagiv M (eds) Proceedings of the 38th ACM SIGPLAN-SIGACT symposium on principles of programming languages, POPL 2011, Austin, TX, USA, January 26–28, 2011. ACM, pp 399–410. https://doi.org/10.1145/1926385.1926431
11. Cook B, Podelski A, Rybalchenko A (2006) Termination proofs for systems code. In: PLDI, pp 415–426
12. Cook B, Podelski A, Rybalchenko A (2011) Proving program termination. Commun ACM 54(5):88–98
13. Cook B, See A, Zuleger F (2013) Ramsey vs. lexicographic termination proving. In: TACAS, pp 47–61
14. Corbet J (2008) Ticket spinlocks. https://lwn.net/Articles/267968/
15. Cousot P, Cousot R (2012) An abstract interpretation framework for termination. In: POPL, pp 245–258
16. Dietsch D, Heizmann M, Langenfeld V, Podelski A (2015) Fairness modulo theory: a new approach to LTL software model checking. In: CAV, lecture notes in computer science, vol 9206. Springer, pp 49–66
17. Fang Y, McMillan KL, Pnueli A, Zuck LD (2006) Liveness by invisible invariants. In: Najm E, Pradat-Peyre J, Donzeau-Gouge V (eds) Formal techniques for networked and distributed systems—FORTE 2006, 26th IFIP WG 6.1 international conference, Paris, France, September 26–29, 2006. Lecture notes in computer science, vol 4229. Springer, pp 356–371. https://doi.org/10.1007/11888116_26

18. Farzan A, Kincaid Z, Podelski A (2016) Proving liveness of parameterized programs. In: LICS. ACM, pp 185–196
19. Ganty P, Genaim S (2013) Proving termination starting from the end. In: Proceedings of the computer aided verification—25th international conference, CAV 2013, Saint Petersburg, Russia, July 13–19, 2013, pp 397–412
20. Ge Y, Moura LD (2009) Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In: International conference on computer aided verification. Springer, pp 306–320
21. Giesl J, Thiemann R, Schneider-Kamp P, Falke S (2004) Automated termination proofs with AProVE. In: RTA, pp 210–220
22. Grebenshchikov S, Lopes NP, Popeea C, Rybalchenko A (2012) Synthesizing software verifiers from proof rules. In: PLDI, pp 405–416
23. Gulwani S, Jain S, Koskinen E (2009) Control-flow refinement and progress invariants for bound analysis. In: PLDI, pp 375–385
24. Harris WR, Lal A, Nori AV, Rajamani SK (2010) Alternation for termination. In: Proceedings of the static analysis—17th international symposium, SAS 2010, Perpignan, France, September 14–16, 2010, pp 304–319
25. Heizmann M, Hoenicke J, Podelski A (2014) Termination analysis by learning terminating programs. In: CAV, lecture notes in computer science, vol 8559. Springer, pp 797–813
26. Hoenicke J, Majumdar R, Podelski A (2017) Thread modularity at many levels: a pearl in compositional verification. In: POPL. ACM, pp 473–485
27. Kesten Y, Pnueli A, Shahar E, Zuck LD (2002) Network invariants in action. In: Proceedings of the 13th international conference on concurrency theory, CONCUR ’02. Springer, Berlin, pp 101–115. http://dl.acm.org/citation.cfm?id=646737.701938
28. Kroening D, Sharygina N, Tsitovich A, Wintersteiger CM (2010) Termination analysis with compositional transition invariants. In: Proceedings of the computer aided verification, 22nd international conference, CAV 2010, Edinburgh, UK, July 15–19, 2010, pp 89–103
29. Lazic M, Konnov I, Widder J, Bloem R (2017) Synthesis of distributed algorithms with parameterized threshold guards. In: 21st international conference on principles of distributed systems, OPODIS 2017, Lisbon, Portugal, December 18–20, 2017, pp 32:1–32:20. https://doi.org/10.4230/LIPIcs.OPODIS.2017.32
30. Lee W, Wang B, Yi K (2012) Termination analysis with algorithmic learning. In: 2012 Proceedings of the computer aided verification—24th international conference, CAV 2012, Berkeley, CA, USA, July 7–13, pp 88–104
31. Manevich R, Dogadov B, Rinetzky N (2016) From shape analysis to termination analysis in linear time. In: CAV (1), lecture notes in computer science, vol 9779. Springer, pp 426–446
32. McMillan KL (2000) A methodology for hardware verification using compositional model checking. Sci Comput Program 37(1–3):279–309. https://doi.org/10.1016/S0167-6423(99)00030-1
33. McMillan KL, Padon O (2018) Deductive verification in decidable fragments with Ivy. In: SAS, lecture notes in computer science, vol 11002. Springer, pp 43–55
34. McMillan KL, Padon O (2020) Ivy: a multi-modal verification tool for distributed algorithms. In: Lahiri SK, Wang C (eds) Proceedings of the computer aided verification—32nd international conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020, Part II, lecture notes in computer science, vol 12225. Springer, pp 190–202. https://doi.org/10.1007/978-3-030-53291-8_12
35. Padon O (2019) Deductive verification of distributed protocols in first-order logic. Ph.D. thesis, Tel Aviv University
36. Padon O, Hoenicke J, Losa G, Podelski A, Sagiv M, Shoham S (2018) Reducing liveness to safety in first-order logic. PACMPL 2(POPL):26:1-26:33. https://doi.org/10.1145/3158114
37. Padon O, Hoenicke J, McMillan KL, Podelski A, Sagiv M, Shoham S (2018) Temporal prophecy for proving temporal properties of infinite-state systems. In: 2018 formal methods in computer-aided design, FMCAD 2018, Austin, Texas, USA, October 30–November 2, 2018, pp 74–84
38. Padon O, McMillan KL, Panda A, Sagiv M, Shoham S (2016) Ivy: safety verification by interactive generalization. In: Proceedings of the 37th ACM SIGPLAN conference on programming language design and implementation, PLDI 2016, Santa Barbara, CA, USA, June 13–17, 2016, pp 614–630
39. Pnueli A, Shahar E (2000) Liveness and acceleration in parameterized verification. In: CAV, lecture notes in computer science, vol 1855. Springer, pp 328–343
40. Podelski A, Rybalchenko A (2004) Transition invariants. In: Proceedings of the 19th IEEE symposium on logic in computer science (LICS 2004), 14–17 July 2004, Turku, Finland, pp 32–41
41. Podelski A, Rybalchenko A (2005) Transition predicate abstraction and fair termination. In: Proceedings of the 32nd ACM SIGPLAN-SIGACT symposium on principles of programming languages, POPL 2005, Long Beach, California, USA, January 12–14, 2005, pp 132–144

42. Srikanth T, Toueg S (1987) Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. Dist Comput 2:80–94
43. Urban C (2013) The abstract domain of segmented ranking functions. In: SAS, lecture notes in computer science, vol 7935. Springer, pp 43–62
44. Urban C, Gurfinkel A, Kahsai T (2016) Synthesizing ranking functions from bits and pieces. In: TACAS, lecture notes in computer science, vol 9636. Springer, pp 54–70
45. Urban C, Miné A (2014) An abstract domain to infer ordinal-valued ranking functions. In: Proceedings of the programming languages and systems—23rd European symposium on programming, ESOP 2014, held as part of the European joint conferences on theory and practice of software, ETAPS 2014, Grenoble, France, April 5–13, 2014, pp 412–431
46. Urban C, Miné A (2014) A decision tree abstract domain for proving conditional termination. In: SAS, lecture notes in computer science, vol 8723. Springer, pp 302–318
47. Urban C, Miné A (2017) Inference of ranking functions for proving temporal properties by abstract interpretation. Comput Lang Syst Struct 47:77–103
48. Widder J, Schmid U (2007) Booting clock synchronization in partially synchronous systems with hybrid process and link failures. Dist Comput 20(2):115–140