# Verifying First-Order Temporal Properties of Infinite-State Systems via Timers and Rankings

Raz Lotan[1,2], Neta Elad[1], Oded Padon[3], and Sharon Shoham[1]

[1] Tel Aviv University, Tel Aviv, Israel
[2] Certora, Tel Aviv, Israel
[3] Weizmann Institute of Science, Rehovot, Israel
lotanraz@gmail.com

**Abstract.** We present a unified deductive verification framework for first-order temporal properties based on well-founded rankings, where verification conditions are discharged using SMT solvers. To that end, we introduce a novel reduction from verification of arbitrary temporal properties to verification of termination. Our reduction augments the system with prophecy timer variables that predict the number of steps along a trace until the next time certain temporal formulas, including the negated property, hold. In contrast to standard tableaux-based reductions, which reduce the problem to fair termination, our reduction does not introduce fairness assumptions. To verify termination of the augmented system, we follow the traditional approach of assigning each state a rank from a well-founded set and showing that the rank decreases in every transition. We leverage the recently proposed formalism of implicit rankings to express and automatically verify the decrease of rank using SMT solvers, even when the rank is not expressible in first-order logic. We extend implicit rankings from finite to infinite domains, enabling verification of more general systems and making them applicable to the augmented systems generated by our reduction, which allows us to exploit the decrease of timers in termination proofs. We evaluate our technique on a range of temporal verification tasks from previous works, giving simple, intuitive proofs for them within our framework.

## 1 Introduction

Several recent works [21, 22, 25, 26, 33] have dealt with verifying temporal properties of transition systems specified in first-order logic. Temporal properties of such systems are naturally specified in First-Order Linear Temporal Logic (FO-LTL). FO-LTL extends standard first-order logic with temporal operators that reason about the behavior of a transition system over time, and allows to specify both safety properties and liveness properties with various fairness assumptions.

Existing approaches for the verification of liveness properties of transition systems given in first-order logic can roughly be divided to those employing a liveness-to-safety reduction or those that use well-founded rankings. The liveness-to-safety reduction of [25] can target any FO-LTL property and is quite powerful

when augmented with temporal prophecy [15, 26]. However, verifying the safety of the resulting system is complicated since it requires reasoning about a monitored copy of the system that tracks repeated states modulo a dynamic finite abstraction. In contrast, well-founded rankings operate at the level of the original system and are therefore more intuitive to use, but current works in this setting [21, 22, 33] only target specific liveness properties and are limited in the kinds of ranking arguments they can capture.

Our goal is to design a verification framework for first-order transition systems that is both intuitive for users and applicable to any temporal property. To this end, we combine two key ideas. First, to obtain a unified approach, we base our framework on a novel reduction from the verification of FO-LTL properties to verification of termination. Our reduction is sound and complete and produces a transition system that is easy to reason about since it is an augmentation of the original transition system with prophecy variables that track violations of the property. Then, verifying termination of the resulting transition system is equivalent to showing that there is no violating trace of the original system. Second, for verifying termination, we adopt and extend the notion of implicit rankings from [21], allowing the user to encode a range of well-founded rankings that may involve the prophecy variables introduced by the reduction. Implicit rankings are provided by the user and verified automatically by an SMT solver.

Our reduction to termination takes a transition system $T$ and a temporal property $\varphi$ and reduces the verification of $T \models \varphi$ to verifying termination of an augmented transition system obtained by a synchronous composition of $T$ with $T_{\neg\varphi}$, the *timer transition system* of $\neg\varphi$. The infinite traces of $T_{\neg\varphi}$ are exactly all the infinite traces that satisfy $\neg\varphi$. The timer transition system resembles standard tableaux constructions, except that it does not have fairness assumptions. As a result, we get a reduction to verification of termination instead of fair termination, which is easier to handle. To construct $T_{\neg\varphi}$ we introduce for each subformula $\psi$ of $\neg\varphi$ a timer variable $t_\psi$ that takes values in $\mathbb{N} \cup \{\infty\}$, and predicts the number of transitions until $\psi$ is next satisfied, or $\infty$ if there is no such number. This behavior is enforced by the transitions of $T_{\neg\varphi}$. The initial states of $T_{\neg\varphi}$ are then defined as those where $t_{\neg\varphi} = 0$ which means that all of their outgoing infinite traces satisfy $\neg\varphi$. It follows that the composition $T \times T_{\neg\varphi}$ is an augmentation of $T$ with timers that contains all infinite traces of $T$ that satisfy $\neg\varphi$. Thus, $T \models \varphi$ holds if and only if the augmented transition system obtained by the composition has no infinite traces, that is, terminates.

To show termination of the transition systems produced by the reduction we employ the method of implicit rankings [21]. Implicit rankings are first-order formulas that capture the decrease of some ranking function between two states, without defining the function explicitly. Implicit rankings are useful for reasoning about ranking functions that are not easily expressible in first-order logic, such as functions that involve unbounded set cardinalities or summations. The constructors of implicit rankings defined in [21] provide an expressive language for well-founded rankings that can be reasoned about in first-order logic, using SMT solvers, but these are defined only for systems that operate over finite,

albeit unbounded, domains. In particular, they cannot leverage timers in rankings, even though their domain is well-founded. To mitigate this, we extend the formalism of implicit rankings to be able to soundly reason over systems with infinite domains. We do so by introducing an explicit soundness condition that allows us, for example, to require that some dynamically updated set is finite in all reachable states or that some relation is well-founded. We further generalize the constructors of [21] to this setting. In particular, this allows us to use the decrease of timers in the ranking used to prove termination of the augmented system produced by our reduction. Importantly, the use of timers in the termination proof relies on their intuitive meaning, and does not require understanding the construction of the augmented system.

We implement our reduction and the constructions of implicit rankings in a deductive verification tool that takes as input a transition system specification and a first-order temporal property, as well as an implicit ranking (given by a composition of constructors) and supporting invariants, which may refer to the timer variables. Our tool uses these ingredients to generate verification conditions that capture the decrease of rank encoded by the implicit rankings in every reachable transition of the system augmented with timers. It then verifies that the system satisfies the property by using an SMT solver to automatically discharge the verification conditions. We evaluate our tool by verifying a set of challenging examples from previous works which include liveness properties of distributed protocols and termination of programs. We give simple intuitive proofs for these examples within our framework.

*Contributions.*

- We introduce a reduction from the verification of arbitrary FO-LTL properties to verification of termination that augments the transition system with prophecy timer variables.
- We generalize the definition of implicit rankings and their constructors from recent work to settings with infinite domains.
- We combine the reduction to termination with implicit rankings to obtain a deductive verification framework where a user constructs an implicit ranking for the augmented system, and the decrease of rank in every step of the augmented system is verified automatically by an SMT solver.
- We implement and evaluate our approach on a set of transition systems and their properties from the literature.

The rest of the paper is organized as follows: Section 2 outlines a motivating example; Section 3 gives background on first-order logic and FO-LTL; Section 4 introduces the reduction from FO-LTL to termination; Section 5 presents the generalized definition of implicit rankings, the proof rule that uses them to prove termination and constructors for implicit rankings; Section 6 details our implementation and evaluation; Section 7 discusses related work and concludes the paper. We defer all proofs to the full version of the paper.

**Full Version.** The full version of the paper is available at [20].

## 2 Running Example

To illustrate our approach we consider the ticket protocol for ensuring mutual exclusion, a variant of Lamport's bakery algorithm [18]. In the ticket protocol, access to the critical section is governed by natural-number tickets. The protocol maintains the next available ticket and the currently serviced ticket, both initialized to zero. Threads start at an idle state and may move to a waiting state by acquiring the next available ticket and in turn increasing it. Once in the waiting state, a thread must wait for its ticket to be the currently serviced ticket to enter the critical section. Immediately after a thread enters the critical section, it transitions back to the idle state while increasing the currently serviced ticket. The control-flow graph of each thread is presented in Fig. 1.

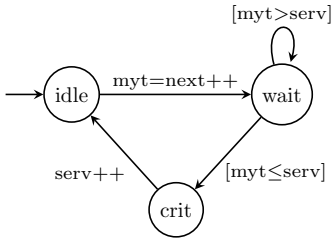


**Fig. 1.** Ticket Protocol

The temporal property we verify is that under fair scheduling, every thread that waits to enter the critical section, eventually does so. We formalize this property in FO-LTL by $\varphi = (\forall x\ \Box\Diamond\text{scheduled}(x)) \rightarrow (\forall x\ \Box(\text{waiting}(x) \rightarrow \Diamond\text{critical}(x)))$ where $\Box$ stands for "globally" and $\Diamond$ stands for "eventually". To verify the property, we need to show that no infinite trace of the system satisfies the negation of $\varphi$. We do so by considering an augmented transition system that restricts the infinite traces of the original system to traces that satisfy the skolemized, negated property $\text{sk}(\neg\varphi) = \forall x.\Box\Diamond\text{scheduled}(x) \wedge \Diamond(\text{waiting}(x_0) \wedge \Box\neg\text{critical}(x_0))$. Verification is then reduced to showing that the augmented transition system has no infinite traces, i.e., that it terminates.

The augmented transition system is constructed automatically by composing the original one with a timer transition system for $\text{sk}(\neg\varphi)$. The general construction is given in Section 5.2. Here, we only illustrate the gist of the construction. The augmented transition system has a timer variable $t_{\text{starved}} \in \mathbb{N} \cup \{\infty\}$ and a timer function $t_{\text{sched}} : \text{Thread} \rightarrow \mathbb{N} \cup \{\infty\}$. The variable $t_{\text{starved}}$ serves as a prophecy variable that predicts the number of steps until $\text{waiting}(x_0) \wedge \Box\neg\text{critical}(x_0)$ will next hold. When $t_{\text{starved}}$ reaches 0 it means that $x_0$ is forever locked out of the critical section. The function $t_{\text{sched}}(x)$ predicts the number of steps until the thread $x$ is next scheduled. The construction ensures that the timers are updated according to this meaning: if their value is positive, it decreases in every step; if it is infinite, it remains infinite; and if it is 0, it is ensured that the formula holds. Because we want to guarantee that $\text{sk}(\neg\varphi)$, the augmented transition system initializes $t_{\text{starved}}$ to some natural number, it then decreases in every step of the system, which, due to the nature of the natural numbers, means it becomes 0 after finitely many steps. Additionally, to guarantee fair scheduling, for every thread $x$, the construction ensures that $t_{\text{sched}}(x)$ is equal to 0 infinitely often. It does so by initializing it to some natural number and resetting it to an arbitrary positive number whenever it reaches 0. Altogether this guarantees that *any* infinite trace of the augmented

system satisfies $\mathrm{sk}(\neg\varphi)$ (without any fairness assumptions). It then remains to prove that the augmented system terminates.

In the remainder of the section we illustrate how timers are leveraged in the termination proof of the augmented system. We emphasize that to incorporate the timers in the proof, the user may rely on their intuitive meaning and need not worry about the construction of the augmented system. Our termination proof is based on a ranking function that maps system states to a rank from some well-founded set that decreases in every transition. However, instead of encoding the function explicitly, we use an extended notion of implicit rankings [21] which encode decrease in the rank between two states, and can be naturally defined using the constructors we provide (see Section 5.2). Due to space constraints, we only present the underlying ranking function. Encoding it as an implicit ranking using our constructors is rather straightforward, given our extension that allows to incorporate timers in implicit rankings despite their infinite domain.

We construct a lexicographic ranking function by considering the possible transitions of the (augmented) system. The first component of the rank is $t_{\mathrm{starved}}$. This timer repeatedly decreases until it reaches 0, from which point on it remains 0, and $x_0$ is starved out of the critical section. The next components of the rank establish that the remainder of the trace from that point on is finite. To that end, we observe that since $x_0$ is in the waiting state indefinitely, its ticket value $\mathrm{myt}(x_0)$ never changes and therefore the difference $\mathrm{myt}(x_0) - \mathrm{serv}$ never increases. When a thread leaves the critical state serv increases and the difference decreases. Therefore, we use the difference as the next component in the rank. Next, when no thread is in the critical section we wait for one to enter. Therefore, the third component in the rank captures whether there is a thread in the critical section. Finally, the scheduled thread may neither be in nor enter the critical section. To ensure decrease of rank in this case, we leverage the timers and introduce a fourth component of the rank that decreases according to $t_{\mathrm{sched}}(x)$ for the thread $x$ that holds the currently serviced ticket and is the next one to enter the critical section. When the resulting rank is encoded with an implicit ranking, the decrease of rank in every (reachable) step of the augmented transition system can be verified automatically by an SMT solver, ensuring termination.

## 3 Preliminaries

*First-Order Logic.* We use many-sorted first-order logic with equality. A signature $\Sigma$ consists of a set of sorts, denoted $\mathrm{sorts}(\Sigma)$, sorted relation, constant and function symbols. Terms are constant symbols, variables or function applications. We use boldface to denote a sequence of variables $\boldsymbol{x}$ or terms $\boldsymbol{t}$. Atomic formulas are $t_1 = t_2$ or relation applications $R(\boldsymbol{t})$. Non-atomic formulas are built using the connectives $\neg, \wedge, \vee, \rightarrow$ and sorted quantifiers $\forall, \exists$. For a formula $\alpha$ (or term), we write $\alpha(\boldsymbol{x})$ to denote that its free variables are contained in $\boldsymbol{x}$, and for a sequence of terms $\boldsymbol{t}$ we write $\alpha(\boldsymbol{t})$ for the simultaneous substitution $\alpha[\boldsymbol{x} \mapsto \boldsymbol{t}]$.

First-order formulas are evaluated over pairs of structures and assignments. A $\Sigma$-structure is a pair $s = (D, I)$ where $D$ maps every sort $\sigma$ to its domain $D(\sigma)$,

which is a non-empty set, and $I$ is an interpretation function for all symbols in $\Sigma$. We denote by $\text{struct}(\Sigma, D)$ the set of all structures over $D$ and by $\text{struct}(\Sigma)$ the class of all $\Sigma$-structures. A sorted assignment $v$ from sorted variables $\boldsymbol{x}$ to $D$ is a function that maps each variable $x_i$ of sort $\sigma$ in $\boldsymbol{x}$ to an element of $D(\sigma)$. We denote the set of all assignments from $\boldsymbol{x}$ to $D$ by $\text{assign}(\boldsymbol{x}, D)$. For two assignments $u, v$ over disjoint sequences of variables $\boldsymbol{x}, \boldsymbol{y}$, we denote by $u{\cdot}v$ the assignment to $\boldsymbol{x}{\cdot}\boldsymbol{y}$ defined by $u{\cdot}v(x_i) = u(x_i)$ and $u{\cdot}v(y_j) = v(y_j)$ for any $i, j$. For a formula $\alpha$ and a pair $(s, v)$ of a structure $s$ and an assignment $v$ to the free variables of $\alpha$ we write $(s, v) \models \alpha$ to denote that $(s, v)$ satisfies $\alpha$.

For a signature $\Sigma$ we denote by $\Sigma'$ a disjoint copy of $\Sigma$ defined by $\Sigma' = \{a' \mid a \in \Sigma\}$. For a sequence of variables $\boldsymbol{x} = (x_i)_{i=1}^m$, we denote by $\boldsymbol{x}'$ the sequence $(x'_i)_{i=1}^m$. For a formula $\alpha$ (or term $t$) over $\Sigma$, we denote by $\alpha'$ the formula over $\Sigma'$ obtained by substituting each symbol $a \in \Sigma$ with $a' \in \Sigma'$. With abuse of notation we sometimes consider a $\Sigma$-structure $s = (D, I)$ as a $\Sigma'$-structure, or vice versa, by setting $I(a') = I(a)$. Similarly, we sometimes consider an assignment to $\boldsymbol{x}$ as an assignment to $\boldsymbol{x}'$, or vice versa, by setting $v(x'_i) = v(x_i)$. For a formula $\tau$ over $\Sigma \uplus \Sigma'$ with free variables $\boldsymbol{x}, \boldsymbol{x}'$ and structures $s, s'$ over a shared domain, we use the notation $(s, v), (s', v') \models \tau$ to denote that $\tau$ is satisfied by the structure-assignment pair where the domain is the shared domain of $s, s'$ and where $\Sigma, \boldsymbol{x}$ are interpreted according to $(s, v)$ and $\Sigma', \boldsymbol{x}'$ are interpreted as in $(s', v')$. If $\tau$ is closed we omit the assignments and write $s, s' \models \tau$.

*Specifying Transition Systems in First-Order Logic.* A transition system over signature $\Sigma$, denoted $\mathcal{T}(\mathcal{S})$, is given by a pair of a first-order specification $\mathcal{T}$ and an intended semantics $\mathcal{S}$. The first-order specification is given by $\mathcal{T} = (\Gamma, \iota, \tau)$ where $\Gamma$ is a closed formula over $\Sigma$ that acts as a background theory, $\iota$ is a closed formula over $\Sigma$ that defines the initial states of the system and $\tau$ is a closed formula over $\Sigma \uplus \Sigma'$ that defines the possible transitions of the system. The intended semantics $\mathcal{S}$ is a class of $\Sigma$-structures. States of $\mathcal{T}(\mathcal{S})$ are given by the $\Sigma$-structures in $\mathcal{S}$ that satisfy $\Gamma$. A trace of $\mathcal{T}(\mathcal{S})$ is given by a sequence of states $\pi = (s_i)_{i=0}^n$ where $n \in \mathbb{N} \cup \{\infty\}$ such that all states in the sequence have the same domain, $s_0 \models \iota$ and for every $i \leq n$ we have $s_i, s_{i+1} \models \tau$. A state $s$ is reachable if there is a finite trace $\pi = (s_i)_{i=0}^n$ for $n \in \mathbb{N}$ such that $s_n = s$. A pair of states $s, s'$ is a reachable transition if $s$ is reachable and $s, s' \models \tau$. We say that $\mathcal{T}(\mathcal{S})$ *terminates* if it has no *infinite* traces.

*First-Order Linear Temporal Logic.* We use FO-LTL to specify temporal properties of transition systems. Given a first-order signature $\Sigma$, FO-LTL formulas are defined similarly to first-order formulas with the addition of temporal operators. For simplicity we only consider the temporal operators 'globally' $\Box$ and 'eventually' $\Diamond$, our approach can be easily extended to the operators 'next' $\bigcirc$ and 'until' $\mathcal{U}$. The semantics of an FO-LTL formula $\psi$ is defined over a pair $(\pi, v)$, where $\pi$ is an *infinite* sequence of $\Sigma$-structures $\pi = (s_i)_{i=0}^\infty$ over a shared domain, and $v$ is an assignment to the free variables of $\psi$. For $\pi = (s_i)_{i=0}^\infty$ and $k \in \mathbb{N}$ we denote $\pi^k = (s_{i+k})_{i=0}^\infty$. The semantics of FO-LTL for $\psi$ and $(\pi, v)$ as above is such that if $\psi$ is atomic we have $\pi, v \models \psi$ if and only if $s_0, v \models \psi$ in

the first-order semantics, $\pi, v \models \Box\psi$ if and only if $\pi^k, v \models \psi$ for every $k \in \mathbb{N}$, $\pi, v \models \Diamond\psi$ if and only if $\pi^k, v \models \psi$ for some $k \in \mathbb{N}$, and the first-order operators are interpreted in the usual way. A temporal property $\varphi$ is given by a closed FO-LTL formula. A transition system $\mathcal{T}(\mathcal{S})$ satisfies a temporal property $\varphi$, denoted $\mathcal{T}(\mathcal{S}) \models \varphi$, if all of its infinite traces satisfy $\varphi$.

## 4 Timers

In this section we present the reduction from verification of FO-LTL properties to verification of termination. First, in Section 4.1, we present the timer transition system of a temporal property $\varphi$, and prove two characteristic facts about it. Then, in Section 4.2, we present the reduction, which uses the timer transition system of the negated temporal property to be verified.

### 4.1 The Timer Transition System

The timer transition system of a temporal property $\varphi$ is a transition system constructed such that (i) all of its infinite traces satisfy $\varphi$ and (ii) it contains all infinite traces that satisfy $\varphi$. We construct the timer transition system by considering a new sort, denoted $\sigma_{\text{time}}$, whose intended semantics are of the extended natural numbers $\mathbb{N} \cup \{\infty\}$, and adding for every subformula $\psi$ a function symbol $t_\psi$ called the *timer* of $\psi$.

Formally, for an FO-LTL property $\varphi$ over signature $\Sigma$ we denote by $A(\varphi)$ the set of formulas that contains all subformulas of $\varphi$, and for every subformula $\Box\psi$, additionally contains $\neg\psi$. The timer transition system $\mathcal{T}_\varphi(\mathcal{S}_{\text{time}})$ is defined over signature $\Sigma_\varphi$ with sorts $\text{sorts}(\Sigma_\varphi) = \text{sorts}(\Sigma) \cup \{\sigma_{\text{time}}\}$ and signature $\Sigma_\varphi = \Sigma \cup \{t_\psi \mid \psi \in A(\varphi)\} \cup \{\leq, 0, \infty, -1\}$, where $t_\psi$ for $\psi(\boldsymbol{x}) \in A(\varphi)$ is a function symbol from the sorts of $\boldsymbol{x}$ to $\sigma_{\text{time}}$; $\leq$ is a binary relation symbol on $\sigma_{\text{time}}$; 0 and $\infty$ are constant symbols of sort $\sigma_{\text{time}}$; and $-1$ is a function symbol from $\sigma_{\text{time}}$ to $\sigma_{\text{time}}$. The timer intended semantics $\mathcal{S}_{\text{time}}$ are given by the class of structures for $\Sigma_\varphi$ where the domain of $\sigma_{\text{time}}$ is $\mathbb{N} \cup \{\infty\}$, with the standard interpretations for $\leq, 0, \infty$ and $-1$. The axioms and transition formula of $\mathcal{T}_\varphi$ are defined such that for any $\psi \in A(\varphi)$, $t_\psi(\boldsymbol{x})$ captures the number of transitions required until $\psi(\boldsymbol{x})$ is satisfied. Finally, the initial states formula ensures that $t_\varphi = 0$ holds in all initial states, which will imply that $\varphi$ is satisfied in every trace.

**Definition 1.** *Let $\varphi$ be an FO-LTL property over signature $\Sigma$, and define $A(\varphi)$, $\Sigma_\varphi$ and $\mathcal{S}_{\text{time}}$ as above. The* timer transition system *of $\varphi$ is a transition system $\mathcal{T}_\varphi(\mathcal{S}_{\text{time}})$ over $\Sigma_\varphi$, where $\mathcal{T}_\varphi = (\Gamma_\varphi, \iota_\varphi, \tau_\varphi)$ is defined by:*

$$\Gamma_\varphi = \bigwedge_{\psi(\boldsymbol{x}) \in A(\varphi)} \forall \boldsymbol{x} \;\; ((t_\psi(\boldsymbol{x}) = 0) \leftrightarrow \zeta_\psi(\boldsymbol{x})) \qquad\qquad \iota_\varphi = (t_\varphi = 0)$$

$$\tau_\varphi = \bigwedge_{\psi(\boldsymbol{x}) \in A(\varphi)} \forall \boldsymbol{x} \left( \begin{array}{l} (0 < t_\psi(\boldsymbol{x}) < \infty \rightarrow t'_\psi(\boldsymbol{x}) = t_\psi(\boldsymbol{x}) - 1) \wedge \\ (t_\psi(\boldsymbol{x}) = \infty \rightarrow t'_\psi(\boldsymbol{x}) = \infty) \wedge \\ (t_\psi(\boldsymbol{x}) = 0) \leftrightarrow \eta_\psi(\boldsymbol{x}) \end{array} \right)$$

**Table 1.** Definitions of $\zeta_\psi, \eta_\psi$ for an FO-LTL formula $\psi(\boldsymbol{x})$

| $\psi(\boldsymbol{x})$ | $\zeta_\psi$ | $\eta_\psi$ |
|---|---|---|
| $R(\boldsymbol{x})$ | $R(\boldsymbol{x})$ | |
| $\neg\rho$ | $t_\rho(\boldsymbol{x}) \neq 0$ | |
| $\rho \circ \chi$ for $\circ \in \{\wedge, \vee, \rightarrow\}$ | $(t_\rho(\boldsymbol{x}) = 0) \circ (t_\chi(\boldsymbol{x}) = 0)$ | |
| $Qy\ \rho$ for $Q \in \{\forall, \exists\}$ | $Qy\ (t_\rho(\boldsymbol{x}{\cdot}y) = 0)$ | |
| $\Diamond\rho$ | $t_\rho(\boldsymbol{x}) < \infty$ | $t_\rho(\boldsymbol{x}) = 0 \vee t'_{\Diamond\rho}(\boldsymbol{x}) = 0$ |
| $\Box\rho$ | $t_{\neg\rho}(\boldsymbol{x}) = \infty$ | $t_\rho(\boldsymbol{x}) = 0 \wedge t'_{\Box\rho}(\boldsymbol{x}) = 0$ |

*where $\zeta_\psi$ and $\eta_\psi$ are defined according to Table 1. If $\eta_\psi$ is undefined, the conjunct that refers to it is omitted from $\tau_\varphi$.*

*Example 1.* Consider $\varphi = \forall x.\Box\Diamond\mathrm{sched}(x)$. The timers in $\mathcal{T}_\varphi$ are $t_{\forall x.\Box\Diamond\mathrm{sched}(x)}$, $t_{\Box\Diamond\mathrm{sched}(x)}(x), t_{\neg\Diamond\mathrm{sched}(x)}(x), t_{\Diamond\mathrm{sched}(x)}(x), t_{\mathrm{sched}(x)}(x)$. In initial states of $\mathcal{T}_\varphi$ we have $t_\varphi = 0$, and so from $\Gamma_\varphi$ we have $\forall x.t_{\Box\Diamond\mathrm{sched}(x)}(x) = 0$ in all initial states. From $\tau_\varphi$ it follows that $\forall x.t_{\Diamond\mathrm{sched}(x)}(x) = 0$ in all reachable states. From the axiom generated for $\Diamond$, for any thread $x$, $t_{\mathrm{sched}(x)}(x) < \infty$ in any reachable state. If $t_{\mathrm{sched}(x)}(x)$, from decrease of timers in $\tau_\varphi$ it follows that this timer decreases to 0 in finitely many steps. Thus, for any thread $x$ we have infinitely often $t_{\mathrm{sched}(x)}(x) = 0$ in any trace, and so fair scheduling is guaranteed. That is, every infinite trace of $\mathcal{T}_\varphi$ satisfies $\varphi$.

*Remark 1.* The construction of the timer transition system resembles tableaux constructions [26, 32]. Instead of using sets of temporal formulas as states, we augment states with timers. In the tableau construction, fairness constraints are used to filter out undesirable traces, such as infinite traces where all states are marked $\Diamond p$ on the basis of some future satisfication of $p$, which never arrives. Instead, in our construction, we rely on the intended semantics of timers to prevent such cases. Since such traces induce infinitely decreasing timer values, the well-foundedness of the natural numbers ensures that they do not exist.

Our first lemma states that for any $\psi \in A(\varphi)$ timers evaluated to 0 capture the satisfaction of the corresponding temporal formula. In $\iota_\varphi$ we require that $t_\varphi = 0$, so it follows that every infinite trace of $\mathcal{T}_\varphi(\mathcal{S}_{\mathrm{time}})$ satisfies $\varphi$.

**Lemma 1.** *Let $\hat{\pi} = (\hat{s}_i)_{i=0}^{\infty}$ be a trace of $\mathcal{T}_\varphi(\mathcal{S}_{\mathrm{time}})$ with domain $D$, $\psi(\boldsymbol{x}) \in A(\varphi)$, $v \in \mathrm{assign}(\boldsymbol{x}, D)$ and $i \in \mathbb{N}$. Then $\hat{s}_i, v \models (t_\psi(\boldsymbol{x}) = 0)$ if and only if $\hat{\pi}^i, v \models \psi(\boldsymbol{x})$.*

The following lemma states that any sequence of states that satisfies $\varphi$ can be augmented to produce a trace of $\mathcal{T}_\varphi(\mathcal{S}_{\mathrm{time}})$. This essentially means that $\mathcal{T}_\varphi(\mathcal{S}_{\mathrm{time}})$ exhibits all the ways in which $\varphi$ can be satisfied. The proof of the lemma works by showing that we can always augment traces with the "natural definition" of timers, which is to evaluate $t_\psi(\boldsymbol{x})$ to the minimal number of transitions that should be taken until the suffix of the sequence satisfies $\psi$, or $\infty$ if none exists.

**Lemma 2.** *Given a sequence of $\Sigma$-structures $\pi = (s_i)_{i=0}^{\infty}$ over a shared domain such that $\pi \models \varphi$, there exists a trace $\hat{\pi} = (\hat{s}_i)_{i=0}^{\infty}$ of $\mathcal{T}_{\varphi}(\mathcal{S}_{\text{time}})$ such that $\hat{\pi}|_{\Sigma} = \pi$.*

### 4.2 The Timer Reduction

The reduction from satisfaction of FO-LTL properties to termination essentially composes $\mathcal{T}(\mathcal{S})$ with the timer transition system of the negated property $\mathcal{T}_{\neg\varphi}(\mathcal{S}_{\text{time}})$. Formally, for two transition systems over signatures $\Sigma_1, \Sigma_2$ with specifications $\mathcal{T}_1 = (\Gamma_1, \iota_1, \tau_1)$, $\mathcal{T}_2 = (\Gamma_2, \iota_2, \tau_2)$ and intended semantics $\mathcal{S}_1, \mathcal{S}_2$ respectively, we define their *product transition system* over $\Sigma_1 \cup \Sigma_2$ where $\Sigma_1 \cup \Sigma_2$ is not necessarily a disjoint union. The product transition system is denoted $(\mathcal{T}_1 \times \mathcal{T}_2)(\mathcal{S}_1 \times \mathcal{S}_2)$ and defined such that $\mathcal{T}_1 \times \mathcal{T}_2 = (\Gamma_1 \wedge \Gamma_2, \iota_1 \wedge \iota_2, \tau_1 \wedge \tau_2)$, and $\mathcal{S}_1 \times \mathcal{S}_2$ contains any $(\Sigma_1 \cup \Sigma_2)$-structure $s$ such that $s|_{\Sigma_1} \in \mathcal{S}_1$ and $s|_{\Sigma_2} \in \mathcal{S}_2$. In our case, $\mathcal{T}(\mathcal{S})$ and $\mathcal{T}_{\neg\varphi}(\mathcal{S}_{\text{time}})$ are defined over signatures $\Sigma, \Sigma_{\varphi}$ respectively such that $\Sigma \subseteq \Sigma_{\varphi}$. Hence, the product transition system is defined over $\Sigma_{\varphi}$ and can be understood as an augmentation of $\mathcal{T}(\mathcal{S})$ with the timer variables and semantics of $\mathcal{T}_{\neg\varphi}(\mathcal{S}_{\text{time}})$. The product system $\mathcal{T}(\mathcal{S}) \times \mathcal{T}_{\neg\varphi}(\mathcal{S}_{\text{time}})$ contains those traces of $\mathcal{T}(\mathcal{S})$ that satisfy $\neg\varphi$, it follows that $\mathcal{T}(\mathcal{S})$ satisfies the property if and only if the product system terminates. The following theorem summarizes the soundness and completeness of the reduction to termination.

**Theorem 1.** *For a transition system $\mathcal{T}(\mathcal{S})$ and an FO-LTL property $\varphi$ we have $\mathcal{T}(\mathcal{S}) \models \varphi$ if and only if $\mathcal{T}(\mathcal{S}) \times \mathcal{T}_{\neg\varphi}(\mathcal{S}_{\text{time}})$ terminates.*

## 5 Rankings

In this section we discuss how we extend and use the framework of implicit rankings from [21] to prove termination of a transition system. Section 5.1 gives the definition of implicit rankings, generalized such that they are parameterized by an explicit soundness condition on reachable states of the system, and shows how implicit rankings are used in termination proofs. Section 5.2 introduces recursive constructions of implicit rankings based on the constructors of [21], extended to produce soundness conditions, and demonstrates how timers are leveraged in the construction of implicit rankings. In Section 5.3 we show how we discharge the proof obligations for a termination proof based on implicit rankings using validity checks in first-order logic.

### 5.1 Verifying Termination with Implicit Rankings

We first introduce our extended definition of *closed* implicit rankings, which are those used for proofs. We then give the definition of implicit rankings, that are not necessarily closed, which can be used for recursive constructions. For the remainder of the section, we fix a first-order signature $\Sigma$. To simplify the notation we assume that $\Sigma$ is single-sorted. The generalization to many-sorted signatures, which we use in practice, is straightforward. A closed implicit ranking is a closed

formula $\tau_>$ that encodes the decrease of some implicit ranking function w.r.t. some implicit order. The formula $\tau_>$ is defined over a double signature $\Sigma \uplus \Sigma'$, where the rank decreases between the pre-state, captured by $\Sigma$, and the post-state, captured by $\Sigma'$. In [21], the implicit order is required to be well-founded. Instead, we relax the requirement and extend the notion of implicit rankings to include a soundness condition $\mathcal{C}$ on states, such that well-foundedness of the implicit order need only be guaranteed for the image of the ranking function on states that satisfy $\mathcal{C}$. We use the soundness condition to specify requirements that are necessary to guarantee well-foundedness of the implicit order but are not directly definable in first-order logic. The soundness conditions we use include the requirement of finiteness of some set of elements and the requirement of well-foundedness of some state relation.

**Definition 2 (Closed Implicit Ranking).** *Given a closed formula $\tau_>$ over $\Sigma \uplus \Sigma'$ and a class of $\Sigma$-structures $\mathcal{C}$, we say that $\tau_>$ is a* closed implicit ranking *with* soundness condition $\mathcal{C}$ *if for any domain $D$, there exist a partially ordered set $(A, <)$ and a function $f\colon \mathrm{struct}(\Sigma, D) \to A$, such that the following hold:*

1. *for every $s, s' \in \mathrm{struct}(\Sigma, D)$ we have $s, s' \models \tau_> \implies f(s) > f(s')$*
2. *$<$ restricted to $\{f(s) \mid s \in \mathcal{C}\}$ is well-founded.*

*We call $(A, <)$ a* ranking range *for $D$ and $f$ a* ranking function *for $D$.*

We note that for using implicit rankings in termination proofs it suffices to require item 1 (or even that $f$ is defined) only for structures in $\mathcal{C}$, but this does not turn out to be helpful for constructing useful implicit rankings.

We use a closed implicit ranking $\tau_>$ with soundness condition $\mathcal{C}$ to show that a transition system $\mathcal{T}(\mathcal{S})$ terminates by showing that all reachable states of $\mathcal{T}(\mathcal{S})$ adhere to the soundness condition $\mathcal{C}$ and that $\tau_>$ is satisfied by all reachable transitions. Termination follows since if the system had an infinite trace, that would imply an infinitely decreasing sequence in a well-founded order. We formalize this in the following theorem:

**Theorem 2.** *If every reachable state $s$ of $\mathcal{T}(\mathcal{S})$ satisfies $s \in \mathcal{C}$ and every reachable transition $(s, s')$ of $\mathcal{T}(\mathcal{S})$ satisfies $s, s' \models \tau_>$ then $\mathcal{T}(\mathcal{S})$ terminates.*

In Section 5.3 we discuss how we verify the requirements of Theorem 2 in practice, before that, we present constructions of implicit rankings. To that end, we first generalize the definition. Similarly to [21], we observe that we can construct implicit rankings by identifying useful primitives and composing them. To allow such recursive constructions which use implicit rankings as building blocks for the construction of complex implicit rankings, [21] generalizes the definition of closed implicit rankings in two ways. First, implicit rankings may have free variables, called *parameters*, in which case the underlying (implicit) ranking function for each domain is defined over pairs of structures and assignments to the free variables. Parameters are useful for constructing implicit rankings that capture aggregations over the domain. Second, implicit rankings in [21] additionally include formulas $\tau_\geq$ that approximate the weak partial order $\leq$ over the ranking

range. This allows recursive constructions to rely on $\leq$ as well. We add a third generalization, which is a formula $\kappa_{\min}$ that approximates whether an element is of minimal rank. This generalization is used to define the soundness conditions of implicit rankings constructed via aggregations over infinite domains.

**Definition 3 (Implicit Ranking).** *An* implicit ranking *with* parameters $\boldsymbol{x}$ *and* soundness condition $\mathcal{C}$ *is a triple* $\mathrm{Rk} = (\tau_>, \tau_\geq, \kappa_{\min})$ *where* $\tau_>, \tau_\geq$ *are formulas over* $\Sigma \uplus \Sigma'$ *with free variables* $\boldsymbol{x}, \boldsymbol{x}', \kappa_{\min}$ *is a formula over* $\Sigma$ *with free variables* $\boldsymbol{x}$ *and* $\mathcal{C}$ *is a class of structures over* $\Sigma$*, such that for any domain* $D$*, there exist a partially ordered set* $(A, <)$ *and a function* $f\colon \mathrm{struct}(\Sigma, D) \times \mathrm{assign}(\boldsymbol{x}, D) \to A$*, such that the following hold:*

1. $(s, v), (s', v') \models \tau_>(\boldsymbol{x}, \boldsymbol{x}') \implies f(s, v) > f(s', v')$
2. $(s, v), (s', v') \models \tau_\geq(\boldsymbol{x}, \boldsymbol{x}') \implies f(s, v) \geq f(s', v')$
3. $(s, v) \models \kappa_{\min} \implies f(s, v)$ is minimal in $(A, <)$
4. $<$ restricted to $\{f(s, v) \mid s \in \mathcal{C}, v \in \mathrm{assign}(\boldsymbol{x}, D)\}$ is well-founded.

*where conditions 1,2,3 range over any* $(s, v), (s', v') \in \mathrm{struct}(\Sigma, D) \times \mathrm{assign}(\boldsymbol{x}, D)$*. We call* $(A, <)$ *a* ranking range *for* $D$ *and* $f$ *a* ranking function *for* $D$*.*

As with closed implicit rankings, we can weaken the requirements by only requiring conditions 1,2,3 for $s, s' \in \mathcal{C}$, but this does not turn out to be useful.

### 5.2 Constructors of Implicit Rankings

We revisit the constructors of [21]. For each, we additionally specify the definition of $\mathcal{C}$ and $\kappa_{\min}$ for the resulting implicit ranking. Moreover, we leverage the soundness condition for implicit rankings as defined in this work to generalize the finite-domain constructors of [21], which are only sound for sorts with finite domains, to allow weaker finiteness assumptions. Due to space constraints, we present only a subset of the constructors that are used to demonstrate the use of timers in ranking and those with non-trivial soundness conditions, all constructors appear in [20].

**Theorem 3.** *All the constructors presented are sound, in the sense that whenever their arguments satisfy their assumptions, they output an implicit ranking with suitable soundness conditions.*

The first constructor we present relies on an existing order over domain elements in the system itself to define an implicit ranking whose underlying ranking function decreases with respect to this order. The order is specified by a relation symbol $\ell(y_1, y_2)$, which is required to satisfy the formula order($\ell$) that specifies that $\ell$ is immutable and defines a strict partial order. The soundness condition of this implicit ranking is that the order defined by $\ell$ is well-founded. We require well-foundedness as a soundness condition as it is not possible to specify in first-order logic. While this may seem like deferring the problem rather than solving it, we are often able to leverage the intended semantics of the transition system, for example the intended semantics of timers $\mathcal{S}_{\mathrm{time}}$, to ensure that the soundness condition holds without explicitly encoding it (see Section 5.3).

**Constructor 1.** *The* position constructor *takes a term $t$ over $\Sigma$ with free variables $\boldsymbol{x}$ and a binary relation symbol $\ell \in \Sigma$, returns an implicit ranking* $\mathrm{Pos}(t, \ell) = (\tau_>, \tau_\geq, \kappa_{\min})$ *with parameters $\boldsymbol{x}$ and soundness condition $\mathcal{C}$ defined by:*

$$\tau_>(\boldsymbol{x}, \boldsymbol{x}') = \mathrm{order}(\ell) \wedge \ell(t'(\boldsymbol{x}'), t(\boldsymbol{x}))$$
$$\tau_\geq(\boldsymbol{x}, \boldsymbol{x}') = \mathrm{order}(\ell) \wedge (\ell(t'(\boldsymbol{x}'), t(\boldsymbol{x})) \vee t'(\boldsymbol{x}') = t(\boldsymbol{x}))$$
$$\kappa_{\min}(\boldsymbol{x}) = \forall y. \neg \ell(y, t(\boldsymbol{x})) \qquad \mathcal{C} = \{s \in \mathrm{struct}(\Sigma) \mid \ell \text{ is well-founded in } s\}$$

*Example 2.* Consider the ticket protocol of Section 2, after having augmented the system with timers. We use the timers in implicit rankings with the position constructor as follows. The relation $\ell$ is the order relation $<$ on $\sigma_{\mathrm{time}}$. Taking $t = t_{\mathrm{waiting}(x_0) \wedge \Box \neg \mathrm{critical}(x_0)}$, we get the closed implicit ranking capturing the decrease of $t_{\mathrm{starved}}$. Taking $t(x) = t_{\Box\Diamond \mathrm{scheduled}(x)}(x)$ we get an implicit rankings with parameter $x$, which needs to be aggregated to get a closed implicit ranking.

The next constructor takes as input an implicit ranking $\mathrm{Rk}^\circ$ and a formula $\alpha$ and returns an implicit ranking that refines the given implicit ranking such that states are ranked based $\mathrm{Rk}^\circ$ only when $\alpha$ is satisfied, and states that do not satisfy $\alpha$ are ranked lower than those that do. It follows that every state that does not satisfy $\alpha$ is minimal according to this ranking. This constructor is useful when we can only guarantee conservation ($\leq$) of $\mathrm{Rk}^\circ$ when $\alpha$ is satisfied.

**Constructor 2.** *The* conditional constructor *takes an implicit ranking* $\mathrm{Rk}^\circ = (\tau^\circ_>, \tau^\circ_\geq, \kappa^\circ_{\min})$ *with parameters $\boldsymbol{x}$ and soundness condition $\mathcal{C}^\circ$, and a formula $\alpha(\boldsymbol{x})$. It returns an implicit ranking* $\mathrm{Cond}(\mathrm{Rk}^\circ, \alpha) = (\tau_>, \tau_\geq, \kappa_{\min})$ *with parameters $\boldsymbol{x}$ and soundness condition $\mathcal{C}$ defined by:*

$$\tau_>(\boldsymbol{x}, \boldsymbol{x}') = (\alpha(\boldsymbol{x}) \wedge \neg\alpha'(\boldsymbol{x}')) \vee (\alpha(\boldsymbol{x}) \wedge \alpha'(\boldsymbol{x}') \wedge \tau^\circ_>(\boldsymbol{x}, \boldsymbol{x}'))$$
$$\tau_\geq(\boldsymbol{x}, \boldsymbol{x}') = \neg\alpha'(\boldsymbol{x}') \vee (\alpha(\boldsymbol{x}) \wedge \alpha'(\boldsymbol{x}') \wedge \tau^\circ_\geq(\boldsymbol{x}, \boldsymbol{x}')) \quad \kappa_{\min}(\boldsymbol{x}) = \neg\alpha(\boldsymbol{x}) \quad \mathcal{C} = \mathcal{C}^\circ$$

*Example 3.* In Ex. 2 we show how to use the timer $t_{\Box\Diamond \mathrm{scheduled}(x)}(x)$ in implicit rankings. In the ticket protocol every thread is scheduled infinitely often so the timer above is not always conserved. Moreover, a thread being scheduled does not necessarily change the state, for example, when it tries to enter the critical section but does not hold serv. For this purpose we want to consider the timer above only for threads that hold the currently served ticket. This is captured by the implicit ranking $\mathrm{Cond}(\mathrm{Pos}(t_{\Box\Diamond \mathrm{scheduled}(x)}(x), <), \mathrm{myt}(x) = \mathrm{serv})$.

We continue by reintroducing the domain-based constructors of [21]. These aggregate rankings that are parameterized by domain elements. In [21] these were defined to be sound only for finite domains. Here, we slightly weaken this assumption and instead introduce it as part of the soundness condition, requiring finiteness only of the set of non-minimal elements in the base ranking. Since minimal values cannot decrease, having finitely-many non-minimal values is sufficient to maintain well-foundedness of the aggregated ranking.

The domain-pointwise constructor lifts an implicit ranking $\mathrm{Rk}^\circ$ with ranking range $(P, <_P)$ to an implicit ranking whose ranking range is the set of functions

$Y \to P$ where $Y$ is possibly infinite. The set $Y \to P$ is ordered by the pointwise ordering: $a <_{\text{pw}} b$ if and only if there exists $y \in Y$ such that $a(y) <_P b(y)$ and for every $y \in Y$ we have $a(y) \leq_P b(y)$. If $<_P$ is well-founded, we can restrict the produced partial order $<_{\text{pw}}$ to be well-founded even if $Y$ is infinite by only considering functions $a \in Y \to P$ that have finitely many $y \in Y$ such that $a(y)$ is non-minimal. This is captured by the soundness condition of the constructor.

**Constructor 3.** *The* domain-pointwise constructor *takes an implicit ranking* $\text{Rk}^\circ = (\tau^\circ_>, \tau^\circ_\geq, \kappa^\circ_{\min})$ *with parameters* $\boldsymbol{x} = \boldsymbol{y}{\cdot}\boldsymbol{z}$ *and soundness condition* $\mathcal{C}^\circ$*. It returns an implicit ranking* $\text{DomPW}(\text{Rk}^\circ, \boldsymbol{y}) = (\tau_>, \tau_\geq, \kappa_{\min})$ *with parameters* $\boldsymbol{z}$ *and soundness condition* $\mathcal{C}$ *defined by:*

$$\tau_>(\boldsymbol{z}, \boldsymbol{z}') = \tau_\geq(\boldsymbol{z}, \boldsymbol{z}') \wedge (\exists \boldsymbol{y}.\tau^\circ_>(\boldsymbol{y}{\cdot}\boldsymbol{z}, \boldsymbol{y}{\cdot}\boldsymbol{z}'))$$
$$\tau_\geq(\boldsymbol{z}, \boldsymbol{z}') = \forall \boldsymbol{y}.\tau^\circ_\geq(\boldsymbol{y}{\cdot}\boldsymbol{z}, \boldsymbol{y}{\cdot}\boldsymbol{z}') \qquad \kappa_{\min}(\boldsymbol{z}) = \forall \boldsymbol{y}.\kappa^\circ_{\min}(\boldsymbol{y}{\cdot}\boldsymbol{z})$$
$$\mathcal{C} = \{s \in \text{struct}(\Sigma) \mid s \in \mathcal{C}^\circ \text{ and } \forall v \in \text{assign}(\boldsymbol{z}).\ \text{finite}_{\boldsymbol{y}}(\neg\kappa^\circ_{\min}(\boldsymbol{y}{\cdot}\boldsymbol{z}), s, v)\}$$

*where* $\text{finite}_{\boldsymbol{y}}(\alpha, s, v) := |\{u \in \text{assign}(\boldsymbol{y}) \mid (s, u{\cdot}v) \models \alpha(\boldsymbol{y}{\cdot}\boldsymbol{z})\}| < \infty$.

*Example 4.* The implicit rankings constructed in Ex. 3 captures the decrease of rank of a scheduling timer for some thread $x$ if it holds the currently served ticket, it thus has $x$ as a parameter. If we aggregate over all such threads with $\text{DomPW}(\text{Cond}(\text{Pos}(t_{\square\lozenge\text{scheduled}(x)}(x), <), \text{myt}(x) = \text{serv}), x)$ we consider the scheduling of all such threads. This pattern is the primary way we utilize timers for implicit rankings, and so we introduce "sugar" for it:

$$\text{TimerRank}(\psi(\boldsymbol{x}), \alpha(\boldsymbol{x})) = \text{DomPW}(\text{Cond}(\text{Pos}(t_\psi(\boldsymbol{x}), <), \alpha(\boldsymbol{x}), \boldsymbol{x}).$$

The final constructor lifts an implicit ranking $\text{Rk}^\circ$ with ranking range $(P, <_P)$ to an implicit ranking whose ranking range is the set of functions $Y \to P$ where $(Y, <_Y)$ is a possibly infinite well-founded set. The set of functions is ordered by the *reverse* lexicographic ordering: $a <_{\text{lex}} b$ if and only if for every $y \in Y$ such that $a(y) \not\leq_P b(y)$ there exists $y_0$ such that $y <_Y y_0$ and $a(y_0) <_P b(y_0)$. Well-foundedness of $<_Y$ along with finiteness of the set of non-minimal elements in $\text{Rk}^\circ$ ensure well-foundedness of the reverse lexicographic ordering. A reverse lexicographic ordering is crucial. To see this, consider for example the standard lexicographic ordering on $\{0,1\}^{\mathbb{N}}$, which is not well-founded since it includes the sequence $(1,0,\ldots), (0,1,0,\ldots), (0,0,1,0,\ldots), \ldots$ that is infinitely decreasing.

**Constructor 4.** *The* domain-lexicographic constructor *takes an implicit ranking* $\text{Rk}^\circ = (\tau^\circ_>, \tau^\circ_\geq, \kappa^\circ_{\min})$ *with parameters* $\boldsymbol{x} = y{\cdot}\boldsymbol{z}$ *and soundness condition* $\mathcal{C}^\circ$*, and a relation* $\ell(y_1, y_2)$ *over* $\Sigma$*. It returns an implicit ranking* $\text{DomLex}(\text{Rk}^\circ, y, \ell) = (\tau_>, \tau_\geq, \kappa_{\min})$ *with parameters* $\boldsymbol{z}$ *and soundness condition* $\mathcal{C}$ *defined by:*

$$\tau_>(\boldsymbol{z}, \boldsymbol{z}') = \tau_\geq(\boldsymbol{z}, \boldsymbol{z}') \wedge (\exists y.\tau^\circ_>(y{\cdot}\boldsymbol{z}, y{\cdot}\boldsymbol{z}'))$$
$$\tau_\geq(\boldsymbol{z}, \boldsymbol{z}') = \text{order}(\ell) \wedge \forall y.(\tau^\circ_\geq(y{\cdot}\boldsymbol{z}, y{\cdot}\boldsymbol{z}') \vee \exists y_0.(\ell(y, y_0) \wedge \tau^\circ_>(y_0{\cdot}\boldsymbol{z}, y_0{\cdot}\boldsymbol{z}')))$$
$$\kappa_{\min}(\boldsymbol{z}) = \forall y.\kappa^\circ_{\min}(y{\cdot}\boldsymbol{z})$$
$$\mathcal{C} = \left\{ s \in \text{struct}(\Sigma) \,\middle|\, \begin{array}{l} s \in \mathcal{C}^\circ, \ell \text{ is well-founded in } s, \\ \forall v \in \text{assign}(\boldsymbol{z}).\ \text{finite}_y(\neg\kappa^\circ_{\min}(y{\cdot}\boldsymbol{z}), s, v) \end{array} \right\}$$

*Example 5.* Consider a program that takes as input an array of natural numbers $c[0, \ldots, n-1]$, and in each iteration, an index $i$ with $c[i] > 0$ is chosen and the operations $c[i] := c[i] - 1; c[i-1] = *$ are performed. This program can be shown to be terminating with the implicit ranking $\mathrm{DomLex}(\mathrm{Pos}(c[i], <_{\mathbb{N}}), i, <_{\mathbb{N}})$.

### 5.3 Validating Decrease of Rank and Soundness Conditions

We now revisit Theorem 2 and show how to verify its two premises in order to prove termination. Fix a signature $\Sigma$, a transition system $\mathcal{T}(\mathcal{S})$ with $\mathcal{T} = (\Gamma, \iota, \tau)$ and a closed implicit ranking $\tau_>$ with soundness condition $\mathcal{C}$. We need to show that any reachable state of $\mathcal{T}(\mathcal{S})$ satisfies the soundness condition $\mathcal{C}$ and any reachable transition satisfies $\tau_>$. To discharge the premises we utilize an inductive invariant $\theta$ as a way to overapproximate the reachable states, for which we need to verify that $\theta$ is satisfied in the initial states and that $\theta$ is preserved by transitions. Then, for the second premise, we need to verify that transitions beginning in a state satisfying $\theta$ satisfy $\tau_>$. These requirements are captured by the validity of the following verification conditions in first-order logic: (i) $\iota \wedge \Gamma \to \theta$ (ii) $\theta \wedge \Gamma \wedge \tau \wedge \Gamma' \to \theta'$ (iii) $\theta \wedge \Gamma \wedge \tau \wedge \Gamma' \to \tau_>$.

As for the soundness condition $\mathcal{C}$, verifying that a certain structure $s$ satisfies $\mathcal{C}$ is not always possible in first-order logic. This is because we use the soundness condition to encode those requirements of the ranking that are not definable in first-order logic. Specifically, the soundness conditions introduced by our constructors are of two types: well-foundedness of a relation and finitneness of a set. These notions are not first-order definable, so we cannot reduce their verification to first-order validity checks. Instead, we employ two complementary strategies to verify the soundness condition: relying on the intended system semantics, or finding sufficient conditions that are expressible in first-order logic.

*Strategy 1.* In some cases the intended system semantics trivially implies the validity of certain soundness conditions: 1. A set of elements of a sort with finite-domain semantics is finite. 2. A partial order on a sort with finite-domain semantics is well-founded. 3. A partial order that is well-founded in the intended semantics of the system, for example, the $<$ relation on a sort with intended semantics of the extended natural numbers (as in timers), is well-founded.

*Strategy 2.* The second strategy is aimed at showing finiteness of a set of elements $S$ in all reachable states, even though the domain of the relevant sort is not finite. The approach is based on [22], with some generalization. The idea is that if there is $m \in \mathbb{N}$ such that initially there are at most $m$ elements in $B$ and any transition adds at most $m$ elements to $B$ then by induction, $B$ is finite in all reachable states. This also implies that any set $A$ such that $A \subseteq B$ in all reachable states is finite. In the soundness conditions we consider, the set $A$ is parametrized by some $\boldsymbol{z}$ and for a given $\boldsymbol{z}$, it is defined as the set of assignments to $\boldsymbol{y}$ that satisfy a formula $\alpha(\boldsymbol{y} \cdot \boldsymbol{z})$, where our goal is to verity finiteness of $A$ for any $\boldsymbol{z}$. We therefore encode the aforementioned idea by considering a formula $\beta(\boldsymbol{y} \cdot \boldsymbol{z})$ used as an inductively-finite over-approximation of $\alpha(\boldsymbol{y} \cdot \boldsymbol{z})$. This results in the following verification conditions, outlined for $m = 1$, for any $\boldsymbol{z}$:

**Table 2.** Evaluation Results. Time is given in seconds, Con. stands for the number of constructors in the implicit ranking, Fin stands for the number of inductively-finite approximations, Inv. stands for the number of conjuncts in the invariant, |Proof| stands for the number of tokens in the entire proof and |L2S Proof| stands for the number of tokens in the entire proof with the liveness-to-safety reduction in the Ivy file.

| Example | Source | Time (s) | Con. | Fin | Inv. | \|Proof\| | \|L2S Proof\| |
|---|---|---|---|---|---|---|---|
| Ticket (Section 2) | [25] | 2 | 6 | 2 | 20 | 256 | 407 |
| Paxos | | 101 | 11 | 1 | 20 | 238 | 521 |
| ABP | | 4 | 31 | 2 | 34 | 539 | 860 |
| HRB-Correct | [3] | 135 | 7 | 0 | 6 | 99 | 446 |
| HRB-Relay | | 56 | 8 | 0 | 17 | 316 | 593 |
| Ackermann | [15] | 0.8 | 5 | 2 | 9 | 211 | 748 |
| LexArray (Ex. 5) | - | 0.08 | 4 | 1 | 2 | 30 | - |
| TimestampedQueue | [22] | 0.5 | 6 | 1 | 2 | 73 | - |
| CascadingQueue | | 1.2 | 9 | 2 | 7 | 175 | - |
| ReorderingQueue | | 2.3 | 9 | 2 | 11 | 220 | - |
| MutexRing | [21] | 0.5 | 8 | 0 | 5 | 93 | - |
| LeaderRing | | 0.3 | 8 | 1 | 4 | 76 | - |
| ToyStabilization | | 0.3 | 5 | 0 | 3 | 69 | - |
| Dijkstra $k$-State | | 64.8 | 6 | 0 | 3 | 213 | - |
| BinaryCounter | | 0.02 | 4 | 0 | 0 | 21 | - |
| SAT-Backtrack | | 0.3 | 9 | 0 | 8 | 157 | - |
| SAT-CDCL | | 5.3 | 8 | 0 | 8 | 148 | - |

(i) $\alpha(\boldsymbol{y}{\cdot}\boldsymbol{z}) \wedge \theta \wedge \Gamma \rightarrow \beta(\boldsymbol{y}{\cdot}\boldsymbol{z})$; (ii) $\iota \wedge \theta \wedge \Gamma \rightarrow \forall \boldsymbol{z}\ \exists \boldsymbol{y}_0\ \forall \boldsymbol{y}\ \beta(\boldsymbol{y}{\cdot}\boldsymbol{z}) \rightarrow \boldsymbol{y} = \boldsymbol{y}_0$; and (iii) $\theta \wedge \Gamma \wedge \tau \wedge \Gamma' \rightarrow \forall \boldsymbol{z}\ \exists \boldsymbol{y}_0\ \forall \boldsymbol{y}\ \beta'(\boldsymbol{y}{\cdot}\boldsymbol{z}) \rightarrow \boldsymbol{y} = \boldsymbol{y}_0 \vee \beta(\boldsymbol{y}{\cdot}\boldsymbol{z})$.

*Example 6.* Continuing with Ex. 4, the soundness condition can be understood as the requirement that in all reachable states there are finitely many threads $x$ for which $\alpha(x) = (\mathrm{myt}(x) = \mathrm{serv})$ holds. If we take $\beta(x) = \neg(\mathrm{idle}(x))$ and an appropriate invariant $\theta$ we can easily verify the verification conditions of strategy 2 above with $m = 1$, and thus validate the soundness condition.

## 6 Implementation and Evaluation

We implemented our approach for verification of temporal properties with timers and implicit rankings in a deductive verification tool. Our tool is implemented in Python, using the Z3 API [23], it is available at [9]. It takes as input a specification of a transition system in first-order logic, along with a temporal property in FO-LTL, an implicit ranking given by a composition of constructors, an inductive

invariant, and, when strategy 2 of Section 5.3 is applied, also inductively-finite approximations, all of which can use the timer variables. The tool computes the timer reduction of Section 4.2, generates the verification conditions of Section 5.3 and discharges them using the Z3 solver. When one of the verification conditions is violated, the tool produces a counterexample in the form of a transition of the augmented system. In the SMT queries, the timers sort $\sigma_{\text{time}}$ is encoded using interpreted integers (with $\infty$ encoded by $-1$). The combination of uninterpreted functions, quantifiers and integers is sometimes difficult for the solver. We therefore implement an optimization where for invariants that do not involve timers we allow to check inductiveness w.r.t. the original system without the timers. Additionally, similarly to [21], we allow hint terms for existential quantifiers which help in queries with quantifier alternations.

We evaluated our tool on a set of examples of transitions systems and their properties from previous works [3,15,21,22,25], with user-provided implicit rankings and invariants. We expand on all examples in [20]. We ran our tool using a Macbook Pro with an Apple M1 Pro CPU, Z3 version 4.13.3.

*Results.* Table 2 summarizes our experimental results. To give a sense of the complexity of the proofs, we report for each example the time required for validating the verification conditions, the number of constructors used in the implicit ranking (where TimerRank is counted as one), the number of inductively-finite approximations needed, and the number of conjuncts in the invariant. We additionally report the total size of the proof, measured by the number of tokens (or terms). Where a proof by the liveness-to-safety method exists, we also report this same metric for the corresponding proof, taken from Ivy [27].

For examples from [21,22], our proofs are similar to the original ones, except that in those works specific liveness proof rules were used while we prove termination of the augmented system. The modifications of the proofs to our general framework are relatively simple, converting premises of the aforementioned rule into instantiations of the TimerRank constructor. For examples that were proven by the liveness-to-safety, our proofs are simpler, as they do not require reasoning about a finite abstraction or a monitored copy of the state. This can be observed in the significant difference in proof size in some examples.

## 7 Related Work

We have presented an approach for verifying temporal properties of systems that operate over unbounded domains, modeled in first-order logic. Many works are aimed at verifying similar systems and properties. Some works take a fully automatic approach that applies to restricted systems [2, 5–7, 10, 11, 14, 16, 28], notably these works do not apply to the setting of systems specified in first-order logic. Other works provide frameworks to design distributed systems and mechanically prove their liveness [4,13,17,19,24,29–31,35] and finally some works provide theoretical frameworks to analyze proof mechanisms for FO-LTL [1, 12, 34]. We turn to discuss the most relevant works, which, similarly to our work, provide semi-automatic methods that rely on first-order logic solvers.

Our work is based on [21] and extends it in two ways. First, while the original work targets only response properties with parameterized fairness assumptions, our work targets any FO-LTL property. Second, the domain-based aggregations of implicit rankings in the original work assume that the relevant domain is finite. In our work we allow domain-based aggregations on infinite domains, as long as the set of non-minimal elements of the aggregated rankings remains finite.

In [22] relational rankings are used to verify temporal properties. Relational rankings are a subset of implicit rankings that are lexicographic compositions of domain-pointwise rankings. Similarly to our work, this work also allows systems with infinite domains, and our strategy 2 in Section 5.3 is inspired by their enforcement of finiteness. Unlike our unified approach, [22] considers different temporal properties by using different proof rules for different types of properties, which is more complex and less general.

The approach of [33] is to semi-automatically synthesize polynomial ranking functions from integer variables in the state of the system. These variables can be from the specification itself, cardinalities of certain sets and fairness variables introduced by the user. These fairness variables are similar to the timers we introduce in Definition 1, and are the inspiration for timers, but they capture only specific types of fairness assumptions and are introduced manually, so they do not have formal guarantees. Like us, they allow the user to denote certain sorts as finitely interpreted, but their treatment of finiteness is less complete. While some examples are shared between the papers, they make some manual manipulations to fairness assumptions in non-trivial ways that significantly simplify the proofs.

In [15, 25, 26] a liveness-to-safety reduction is employed to reduce the verification of temporal properties to safety verification. The reduction performs a dynamic finite abstraction of the system. One then proves acyclicity of the abstracted system by providing an invariant for a monitored system where the current state is compared to a saved state modulo the abstraction. The liveness-to-safety reduction is sound but not complete, and requires employing temporal prophecy to extend its power. Moreover, writing invariants for the augmented system may be more complicated and less intuitive than giving implicit rankings.

**Acknowledgement** We thank the anonymous reviewers and Eden Frenkel for helpful comments. The research leading to these results has received funding from the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant agreement No [759102-SVIS]). This research was partially supported by the Israeli Science Foundation (ISF) grant No. 2117/23, by a research grant from the Center for New Scientists at the Weizmann Institute of Science and by a grant from the Azrieli Foundation.

**Data-Availability Statement** The paper is accompanied by an artifact containing the software and data required to reproduce the results. A snapshot of the artifact corresponding to the version used in this paper is available at [9]. For reuse and future development, the source code is available at [8].

## References


1. Abadi, M.: The power of temporal proofs. Theor. Comput. Sci. **65**(1), 35–83 (1989), https://doi.org/10.1016/0304-3975(89)90138-2
2. Ben-Amram, A.M., Lee, C.S.: Ranking functions for size-change termination II. Log. Methods Comput. Sci. **5**(2) (2009), http://arxiv.org/abs/0903.4382
3. Berkovits, I., Lazic, M., Losa, G., Padon, O., Shoham, S.: Verification of threshold-based distributed algorithms by decomposition to decidable logics. In: Computer Aided Verification - 31st International Conference, CAV 2019 (2019), https://doi.org/10.1007/978-3-030-25543-5_15
4. Brunel, J., Chemouil, D., Tawa, J.: Analyzing the fundamental liveness property of the chord protocol. In: Formal Methods in Computer Aided Design, FMCAD 2018 (2018), https://doi.org/10.23919/FMCAD.2018.8603001
5. Chen, Y., Heizmann, M., Lengál, O., Li, Y., Tsai, M., Turrini, A., Zhang, L.: Advanced automata-based algorithms for program termination checking. In: Programming Language Design and Implementation, PLDI 2018 (2018), https://doi.org/10.1145/3192366.3192405
6. Cimatti, A., Griggio, A., Magnago, E.: LTL falsification in infinite-state systems. Information and Computation (2022), https://www.sciencedirect.com/science/article/pii/S0890540122001328
7. Daniel, J., Cimatti, A., Griggio, A., Tonetta, S., Mover, S.: Infinite-state liveness-to-safety via implicit abstraction and well-founded relations. In: Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016 (2016), https://doi.org/10.1007/978-3-319-41528-4_15
8. Elad, N., Lotan, R.: implicit-rankings-timers, https://github.com/neta-elad/implicit-rankings-timers
9. Elad, N., Lotan, R.: Verifying first-order temporal properties of infinite states systems via timers and rankings (artifact) (Jan 2026), https://doi.org/10.5281/zenodo.18157346
10. Fang, Y., Piterman, N., Pnueli, A., Zuck, L.D.: Liveness with invisible ranking. Int. J. Softw. Tools Technol. Transf. **8**(3), 261–279 (2006). https://doi.org/10.1007/S10009-005-0193-X
11. Farzan, A., Kincaid, Z., Podelski, A.: Proving liveness of parameterized programs. In: Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16 (2016). https://doi.org/10.1145/2933575.2935310
12. Geatti, L., Gianola, A., Gigante, N.: First-order automata. In: Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence (2025), https://doi.org/10.1609/aaai.v39i14.33638
13. Hawblitzel, C., Howell, J., Kapritsos, M., Lorch, J.R., Parno, B., Roberts, M.L., Setty, S.T.V., Zill, B.: Ironfleet: proving safety and liveness of practical distributed systems. Commun. ACM **60**(7), 83–92 (2017), https://doi.org/10.1145/3068608
14. Herrmann, R., Rümmer, P.: A new approach for showing termination of parameterized transition systems. In: Implementation and Application of Automata (2026), https://doi.org/10.1007/978-3-032-02602-6_14
15. Hoenicke, J., Padon, O., Shoham, S.: On the Power of Temporal Prophecy, pp. 40–53. Springer Nature Switzerland, Cham (2026), https://doi.org/10.1007/978-3-032-13711-1_3
16. Hong, C., Lin, A.W.: Regular abstractions for array systems. Proc. ACM Program. Lang. **8**(POPL), 638–666 (2024), https://doi.org/10.1145/3632864

17. Ioannidis, E., Zakowski, Y., Zdancewic, S., Angel, S.: Structural temporal logic for mechanized program verification (2025), https://arxiv.org/abs/2410.14906
18. Lamport, L.: A new solution of dijkstra's concurrent programming problem. Commun. ACM **17**(8), 453–455 (1974), https://doi.org/10.1145/361082.361093
19. Leung, D., Zeldovich, N., Kaashoek, F.: Shipwright: Proving liveness of distributed systems with byzantine participants (2025), https://arxiv.org/abs/2507.14080
20. Lotan, R., Elad, N., Padon, O., Shoham, S.: Verifying first-order temporal properties of infinite-state systems via timers and rankings (2026), https://arxiv.org/abs/2601.13325
21. Lotan, R., Shoham, S.: Implicit rankings for verifying liveness properties in first-order logic. In: Tools and Algorithms for the Construction and Analysis of Systems TACAS 2025 (2025). https://doi.org/10.1007/978-3-031-90643-5_20
22. McMillan, K.L.: Toward liveness proofs at scale. In: Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14681, pp. 255–276. Springer (2024). https://doi.org/10.1007/978-3-031-65627-9_13
23. de Moura, L.M., Bjørner, N.S.: Z3: an efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, (2008). https://doi.org/10.1007/978-3-540-78800-3_24
24. Ouyang, L., Sun, X., Tang, R., Huang, Y., Jivrajani, M., Ma, X., Xu, T.: Multi-grained specifications for distributed system model checking and verification. In: Proceedings of the Twentieth European Conference on Computer Systems. EuroSys '25 (2025). https://doi.org/10.1145/3689031.3696069
25. Padon, O., Hoenicke, J., Losa, G., Podelski, A., Sagiv, M., Shoham, S.: Reducing liveness to safety in first-order logic. Proc. ACM Program. Lang. **2**(POPL), 26:1–26:33 (2018). https://doi.org/10.1145/3158114
26. Padon, O., Hoenicke, J., McMillan, K.L., Podelski, A., Sagiv, M., Shoham, S.: Temporal prophecy for proving temporal properties of infinite-state systems. Formal Methods Syst. Des. **57**(2), 246–269 (2021). https://doi.org/10.1007/S10703-021-00377-1
27. Padon, O., McMillan, K.L., Panda, A., Sagiv, M., Shoham, S.: Ivy: Safety verification by interactive generalization. In: PLDI '16: Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 614–630 (Jun 2016). https://doi.org/10.1145/2908080.2908118
28. Podelski, A., Rybalchenko, A.: Transition invariants. In: 19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings. pp. 32–41. IEEE Computer Society (2004). https://doi.org/10.1109/LICS.2004.1319598
29. Qiu, L., Kim, Y., Shin, J., Kim, J., Honoré, W., Shao, Z.: Lido: Linearizable byzantine distributed objects with refinement-based liveness proofs. Proc. ACM Program. Lang. **8**(PLDI), 1140–1164 (2024), https://doi.org/10.1145/3656423
30. Qiu, L., Xiao, J., Shin, J.Y., Shao, Z.: Lido-dag: A framework for verifying safety and liveness of dag-based consensus protocols. Proc. ACM Program. Lang. **9**(PLDI) (Jun 2025), https://doi.org/10.1145/3729306
31. Sun, X., Ma, W., Gu, J.T., Ma, Z., Chajed, T., Howell, J., Lattuada, A., Padon, O., Suresh, L., Szekeres, A., Xu, T.: Anvil: Verifying liveness of cluster management controllers. In: 18th USENIX Symposium on Operating Systems Design and Implementation (2024), https://www.usenix.org/conference/osdi24/presentation/sun-xudong

32. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Logics for Concurrency - Structure versus Automata (1995). https://doi.org/10.1007/3-540-60915-6_6
33. Yao, J., Tao, R., Gu, R., Nieh, J.: Mostly automated verification of liveness properties for distributed protocols with ranking functions. Proc. ACM Program. Lang. **8**(POPL), 1028–1059 (2024). https://doi.org/10.1145/3632877
34. Zhang, W.: First order büchi automata and their application to verification of LTL specifications. J. Log. Algebraic Methods Program. **142**, 101021 (2025). https://doi.org/10.1016/J.JLAMP.2024.101021
35. Zhao, Q., Pîrlea, G., Grzeszkiewicz, K., Gilbert, S., Sergey, I.: Compositional verification of composite byzantine protocols. In: Conference on Computer and Communications Security. CCS '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3658644.3690355