Fairness for Infinite-State Systems

Byron Cook[1], Heidy Khlaaf[1], and Nir Piterman[2]

[1] University College London, London, UK
[2] University of Leicester, Leicester, UK

Abstract. In this paper we introduce the first known tool for symbolically proving *fair*-CTL properties of (infinite-state) integer programs. Our solution is based on a reduction to existing techniques for fairness-free CTL model checking via the use of infinite non-deterministic branching to symbolically partition fair from unfair executions. We show the viability of our approach in practice using examples drawn from device drivers and algorithms utilizing shared resources.

1 Introduction

In model checking, fairness allows us to bridge between linear-time (*a.k.a.* trace-based) and branching-time (*a.k.a.* state-based) reasoning. Fairness is crucial, for example, to Vardi & Wolper's automata-theoretic technique for LTL verification [25]. Furthermore, when proving state-based CTL properties, we must often use fairness to model trace-based assumptions about the environment both in a sequential setting, and when reasoning about concurrent environments, where fairness is used to abstract away the scheduler.

In this paper we introduce the first-known fair-CTL model checking technique for (infinite-state) integer programs. Our solution reduces fair-CTL to fairness-free CTL using prophecy variables to encode a partition of fair from unfair paths. Cognoscenti may at first find this result surprising. It is well known that fair termination of Turing machines cannot be reduced to termination of Turing machines. The former is Σ_1^1-complete and the latter is RE-complete [18].[3] For similar reasons fair-CTL model checking of Turing machines cannot be reduced to CTL model checking of Turing machines. The key to our reduction is the use of infinite non-deterministic branching when model checking fairness-free CTL. As a consequence, in the context of infinite branching, fair and fairness-free CTL are equally difficult (and similarly for termination).

Motivation. Current techniques for model checking CTL properties provide no support for verification of fair-CTL, thus excluding a large set of branching-time liveness properties necessitating fairness. These properties are often imperative to verifying the liveness of systems such as Windows kernel APIs that acquire resources and APIs that release resources. Below are properties which can be expressed in fair-CTL, but not CTL nor LTL. We write these properties in CTL*,

[3] Sometimes termination refers to *universal termination*, which entails termination for *all* possible inputs. This is a harder problem and is co-RE$^{\mathrm{RE}}$-complete.

a superset of both CTL and LTL[4]. For brevity, we write Ω for $\mathsf{GF}p{\rightarrow}\mathsf{GF}q$. A state property is indicated by φ (i.e., a combination of assertions on the states of the program) and p and q are subsets of program states, constituting our fairness requirement (infinitely often p implies infinitely often q).

The property $\mathsf{E}[\Omega \wedge \mathsf{G}\varphi]$ generalizes fair non-termination, that is, there exists an infinite fair computation all of whose states satisfy the property φ. The property $\mathsf{A}\big[\Omega \rightarrow \mathsf{G}[\varphi_1 \rightarrow \mathsf{A}(\Omega \rightarrow \mathsf{F}\varphi_2)]\big]$ indicates that on every fair path, every φ_1 state is later followed by a φ_2 state. We will later verify it for a Windows device driver, indicating that a lock will always eventually be released in the case that a call to a lock occurs, provided that whenever we continue to call a Windows API repeatedly, it will eventually return a desired value (fairness). Similarly, $\mathsf{A}\big[\Omega \rightarrow \mathsf{G}[\varphi_1 \rightarrow \mathsf{A}(\Omega \rightarrow \mathsf{FE}(\Omega \wedge \mathsf{G}\varphi_2))]\big]$ dictates that on every fair path whenever a φ_1 state is reached, on all possible futures there is a state which is a possible fair future and φ_2 is always satisfied. For example, one may wish to verify that there will be a possible active fair continuation of a server, and that it will continue to effectively serve if sockets are successfully opened.[5]

Furthermore, fair-CTL model checking is rudimentary to the well known technique of verifying LTL in the finite-state setting [25]. A fair-CTL model checker for infinite-state systems would thus enable us to implement the automata-theoretic approach to linear-time model checking by reducing it to fair-CTL model checking as is done in the finite-state setting.

Fairness is also crucial to the verification of concurrent programs, as well-established techniques such as [7] reduce concurrent liveness verification to a sequential verification task. Thread-modular reductions of concurrent to sequential programs often require a concept of fairness when the resulting sequential proof obligation is a progress property such as wait-freedom, lock-freedom, or obstruction-freedom. Moreover, obstruction freedom cannot be expressed in LTL without additional assumptions. With our technique we can build tools for automatically proving these sequential reductions using fair-CTL model checking.

Related Work. Support for fairness in finite and other decidable settings has been well studied. Tools for these settings (*e.g.* NuSMV for finite state systems [5, 6], Moped and PuMoc for pushdown automata [23, 24], Prism for probabilistic timed automata [19], and Uppaal for timed automata [15]) provide support for fairness constraints. Proof systems for the verification of temporal properties of fair systems (e.g., [3], [21]) also exist. However, such systems require users to construct auxiliary assertions and participate in the proof process.

Contrarily, we seek to automatically verify the undecidable general class of (infinite-state) integer programs supporting both control-sensitive and numerical properties. Additionally, some of these tools do not fully support CTL model checking, as they do not reliably support mixtures of nested universal/existential path quantifiers, etc. The tools which consider full CTL and the general class of

[4] These properties expressed in terms of the fair path quantifiers E_f and A_f are $\mathsf{E}_f\,\mathsf{G}\varphi, \mathsf{A}_f\,\mathsf{G}(\varphi_1 \rightarrow \mathsf{A}_f\,\mathsf{F}\varphi_2)$, and $\mathsf{A}_f\,\mathsf{G}(\varphi_1 \rightarrow \mathsf{A}_f\,\mathsf{F}\,\mathsf{E}_f\,\mathsf{G}\varphi_2)$, respectively.

[5] Notice that our definition of fair CTL considers finite paths. Thus, all path quantifications above range over finite paths as well.

integer programs as we do are [2], [10], and [12]. However, these tools provide no support for verifying fair-CTL.

When we consider the general class of integer programs, the use of infinite nondeterminism to encode fairness policies has been previously utilized by Olderog *et al.* [1]. However, they do not rely on nondeterminism alone but require refinement of the introduced nondeterminism to derive concrete schedulers which enforce a given fairness policy. Thus, their technique relies on the ability to force the occurrence of fair events whenever needed by the reduction. We support general fairness constraints, rather than just fair scheduling. The ability to force the occurrence of fair events is too strong for our needs. Indeed, in the context of model checking we rely on the program continuing a normal execution until the "natural" fulfillment of the fairness constraint.

An analysis of fair discrete systems which separates reasoning pertaining to fairness and well-foundedness through the use of inductive transition invariants was introduced in [20]. Their strategy is the basis of the support for fairness added to Terminator [8]. However, this approach relies on the computation of transition invariants [22], whereas our approach does not. We have recently shown that, in practice, state-based techniques that circumvent the computation of transition invariants perform significantly better [14]. Additionally, a technique utilized to reduce LTL model checking to fairness-free CTL model checking introduced by [11] is largely incomplete, as it does not sufficiently determinize all possible branching traces. Note that these methodologies are used to verify fairness and liveness constraints expressible within linear temporal logic, and are thus not applicable to verify fair branching-time logic or branching-time logic. Indeed, this was part of our motivation for studying alternative approaches to model checking with fairness.

2 Preliminaries

Transition systems. A transition system is $M = (S, S_0, R, L)$, where S is a countable set of states, $S_0 \subseteq S$ a set of initial states, $R \subseteq S \times S$ a transition relation, and $L : S \to 2^{AP}$ a labeling function associating a set of propositions with every state $s \in S$. A *trace* or a *path* of a transition system is either a finite or infinite sequence of states. The set of infinite traces starting at $s \in S$, denoted by $\Pi_\infty(s)$, is the set of sequences $(s_0, s_1, \ldots)$ such that $s_0 = s$ and $\forall i \geq 0.\ (s_i, s_{i+1}) \in R$. The set of finite traces starting at $s \in S$, denoted by $\Pi_f(s)$, is the set of sequences $(s_0, s_1, \ldots, s_j)$ such that $s_0 = s$, $j \geq 0$, $\forall i < j.\ (s_i, s_{i+1}) \in R$, and $\forall s \in S.\ (s_j, s) \notin R$. Finally, the set of maximal traces starting at s, denoted by $\Pi_m(s)$, is the set $\Pi_\infty(s) \cup \Pi_f(s)$. For a path π, we denote the length of said path by $|\pi|$, which is ω in case that π is infinite.

Computation tree logic (CTL). We are interested in verifying state-based properties in computation tree logic (CTL). Our definition of CTL differs slightly from previous work, as it takes into account finite (maximal) paths. This semantics allows us to specify properties such as termination without requiring special

$$\frac{\alpha(s)}{M, s \models_m \alpha} \quad \frac{\neg\alpha(s)}{M, s \models_m \neg\alpha}$$

$$\frac{M, s \models \varphi_1 \quad M, s \models \varphi_2}{M, s \models_m \varphi_1 \wedge \varphi_2} \quad \frac{M, s \models \varphi_1 \ \vee \ M, s \models \varphi_2}{M, s \models_m \varphi_1 \vee \varphi_2}$$

$$\frac{\forall \pi = (s_0, s_1, ...) \in \Pi_m(s).\ M, s_1 \models \varphi}{M, s \models_m \mathsf{AX}\varphi} \quad \frac{\exists \pi = (s_0, s_1, ...) \in \Pi_m(s).\ M, s_1 \models \varphi}{M, s \models_m \mathsf{EX}\varphi}$$

$$\frac{\begin{array}{c}\forall \pi = (s_0, s_1, ...) \in \Pi_m(s).\ (\forall i \in [0, |\pi|).\ M, s_i \models \varphi_1) \vee \\ (\exists j \in [0, |\pi|).\ M, s_j \models \varphi_2 \wedge \forall i \in [0, j).\ M, s_i \models \varphi_1)\end{array}}{M, s \models_m \mathsf{A}[\varphi_1 \mathsf{W} \varphi_2]} \quad \frac{\begin{array}{c}\forall \pi = (s_0, s_1, ...) \in \Pi_m(s). \\ (\exists j \in [0, |\pi|).\ M, s_j \models \varphi)\end{array}}{M, s \models_m \mathsf{AF}\varphi}$$

$$\frac{\begin{array}{c}\exists \pi = (s_0, s_1, ...) \in \Pi_m(s). \\ (\exists j \in [0, |\pi|).\ M, s_j \models \varphi_2 \wedge \forall i \in [0, j).\ M, s_i \models \varphi_1)\end{array}}{M, s \models_m \mathsf{E}[\varphi_1 \mathsf{U} \varphi_2]} \quad \frac{\begin{array}{c}\exists \pi = (s_0, s_1, ...) \in \Pi_m(s). \\ (\forall i \in [0, |\pi|).\ M, s_i \models \varphi)\end{array}}{M, s \models_m \mathsf{EG}\varphi}$$

Fig. 1. Semantics of CTL: $\models_m$

atomic propositions to hold at program exit points, as proposed by Cook *et al.* in [13], and to reason about a transformation that introduces many finite paths.

A CTL formula is of the form:

$$\varphi ::= \alpha \mid \neg\alpha \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathsf{AX}\varphi \mid \mathsf{AF}\varphi \mid \mathsf{A}[\varphi \mathsf{W} \varphi] \mid \mathsf{EX}\varphi \mid \mathsf{EG}\varphi \mid \mathsf{E}[\varphi \mathsf{U} \varphi],$$

where $\alpha \in AP$ is an atomic proposition. We assume that formulae are written in negation normal form, in which negation only occurs next to atomic propositions. We introduce $\mathsf{AG}, \mathsf{AU}, \mathsf{EF}$, and EW as syntactic sugar as usual. A formula is in ACTL if it uses only universal operators, i.e., AX, AW, AF, AU, or AG.

Fig. 1 defines when a CTL property φ holds in a state $s \in S$ of a transition system M. We say that φ holds in M, denoted $M \models_m \varphi$, if $\forall s \in S_0.\ M, s \models_m \varphi$.

Fair CTL. For a transition system M, a fairness condition is $\Omega = (p, q)$, where $p, q \subseteq S$. When fairness is part of the transition system we denote it as $M = (S, S_0, R, L, \Omega)$. We still include Ω as a separate component in transformations and algorithms for emphasis. We freely confuse between assertions over program variables and sets of states that satisfy them. An infinite path π is unfair under Ω if states from p occur infinitely often along π but states from q occur finitely often. Otherwise, π is fair. The condition Ω denotes a strong fairness constraint. Weak fairness constraints can be trivially expressed by $\Omega = (\mathsf{true}, q)$, that is, states from q must occur infinitely often. Equivalently, π is fair if it satisfies the LTL formula $\pi \models (\mathsf{GF}p \rightarrow \mathsf{GF}q)$. For a transition system $M = (S, S_0, R, L, \Omega)$, an infinite path π, we denote $M, \pi \models \Omega$ if π is fair [17]. We consider strong fairness with one pair of sets of states. Extending our results to strong fairness over multiple pairs is simple and omitted for clarity of exposition.

For a transition system M and a CTL property φ, the definition of when φ holds in a state $s \in S$ is defined as in Fig. 1 except that $\Pi_m(s)$ is redefined to be $\Pi_f \cup \{\pi \in \Pi_\infty \mid M, \pi \models \Omega\}$. We use the notation $\models_{\Omega+}$ when expressing fairness,

$$\textsc{Fair}((S, S_0, R, L), (p, q)) \triangleq (S_\Omega, S^0_\Omega, R_\Omega, L_\Omega) \text{ where}$$

$$\begin{array}{rcl} S_\Omega & = & S \times \mathbb{N} \\ R_\Omega & = & \{((s,n),(s',n')) \mid (s,s') \in R\} \wedge \begin{pmatrix} (\neg p \wedge n' \leq n) \vee \\ (p \wedge n' < n) \vee \\ q \end{pmatrix} \\ S^0_\Omega & = & S^0 \times \mathbb{N} \\ L_\Omega(s,n) & = & L(s) \end{array}$$

Fig. 2. Fair takes a system (S, S_0, R, L) and a fairness constraint (p, q) where $p, q \subseteq S$, and returns a new system $(S_\Omega, S^0_\Omega, R_\Omega, L_\Omega)$. Note that $n \geq 0$ is implicit, as $n \in \mathbb{N}$.

that is, we say that φ holds in M, denoted by $M \models_{\Omega_+} \varphi$, if $\forall s \in S_0.M, s \models_{\Omega_+} \varphi$. When clear from the context, we may omit M and simply write $s \models_{\Omega_+} \varphi$ or $s \models_m \varphi$.

3 Fair-CTL Model Checking

In this section we present a procedure for reducing fair-CTL model checking to CTL model checking. The procedure builds on a transformation of infinite-state programs by adding a prophecy variable that truncates unfair paths. We start by presenting the transformation, followed by a program's adaptation for using said transformation, and subsequently the model-checking procedure.

In Fig. 2, we propose a reduction $\textsc{Fair}(M, \Omega)$ that encodes an instantiation of the fairness constraint within a transition system. When given a transition system (S, S_0, R, L, Ω), where $\Omega = (p, q)$ is a strong-fairness constraint, $\textsc{Fair}(M, \Omega)$ returns a new transition system (without fairness) that, through the use of a prophecy variable n, infers all possible paths that satisfy the fairness constraint, while avoiding all paths violating the fairness policy. Intuitively, n is decreased whenever a transition imposing $p \wedge n' < n$ is taken. Since $n \in \mathbb{N}$, n cannot decrease infinitely often, thus enforcing the eventual invalidation of the transition $p \wedge n' < n$. Therefore, R_Ω would only allow a transition to proceed if q holds or $\neg p \wedge n' \leq n$ holds. That is, either q occurs infinitely often or p will occur finitely often. Note that a q-transition imposes no constraints on n', which effectively resets n' to an arbitrary value. Recall that extending our results to multiple fairness constraints is simple and omitted for clarity of exposition.

The conversion of M with fairness constraint Ω to $\textsc{Fair}(M, \Omega)$ involves the truncation of paths due to the wrong estimation of the number of p-s until q. This means that $\textsc{Fair}(M, \Omega)$ can include (maximal) finite paths that are prefixes of unfair infinite paths. So when model checking CTL we have to ensure that these paths do not interfere with the validity of our model checking procedure. Hence, we distinguish between maximal (finite) paths that occur in M and those introduced by our reduction. We do this by adding a proposition t to mark all original "valid" termination states prior to the reduction in Fig. 2 and by adjusting the CTL specification. These are presented in Section 3.3. We first provide high-level understanding of our approach through an example.

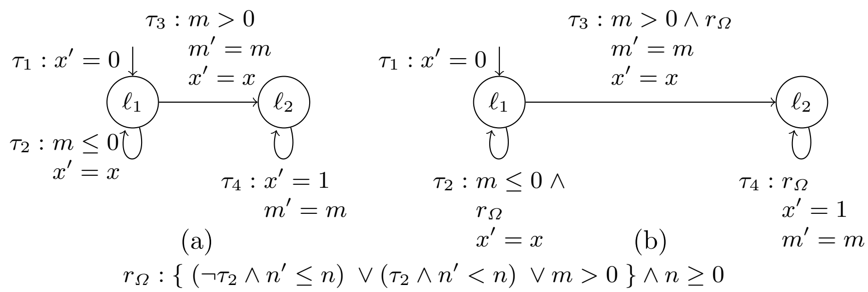


Fig. 3. Reducing a transition system with the fair CTL property $\mathsf{AG}(x = 0 \to \mathsf{AF}(x = 1))$ and the fairness constraint $\mathsf{GF}\ \tau_2 \to \mathsf{GF}\ m > 0$. The original transition system is represented in (a), followed by the application of our fairness reduction in (b).

3.1 Illustrative Example

Consider the example in Fig. 3 for the fair CTL property $\mathsf{AG}(x = 0 \to \mathsf{AF}(x = 1))$ and the fairness constraint $\mathsf{GF}\ \tau_2 \to \mathsf{GF}\ m > 0$ for the initial transition system introduced in (a). We demonstrate the resulting transformation for this infinite-state program, which allows us to reduce fair model checking to model checking. By applying $\textsc{Fair}(M, \Omega)$ from Fig. 2, we obtain (b) where each original transition, τ_2, τ_3, and τ_4, are adjoined with restrictions such that $\{(\neg\tau_2 \wedge n' \leq n) \vee (\tau_2 \wedge n' < n) \vee m > 0 \} \wedge n \geq 0$ holds. That is, we wish to restrict our transition relations such that if τ_2 is visited infinitely often, then the variable m must be positive infinitely often. In τ_2, the unconstrained variable m indicates that the variable m is being assigned to a nondeterministic value, thus with every iteration of the loop, m acquires a new value. In the original transition system, τ_2 can be taken infinitely often given said non-determinism, however in (b), such a case is not possible. The transition τ_2 in (b) now requires that n be decreased on every iteration. Since $n \in \mathbb{N}$, n cannot be decreased infinitely often, causing the eventual restriction to the transition τ_2. Such an incidence is categorized as a finite path that is a prefix of some unfair infinite paths. As previously mentioned, we will later discuss how such paths are disregarded. This leaves only paths where the prophecy variable "guessed" correctly. That is, it prophesized a value such that τ_3 is reached, thus allowing our property to hold.

3.2 Prefixes of Infinite Paths

We explain how to distinguish between maximal (finite) paths that occur in M, and those that are prefixes of unfair infinite paths introduced by our reduction. Consider a transition system $M = (S, S_0, R, L, \Omega)$, where $\Omega = (p, q)$, and let φ be a CTL formula. Let t be an atomic proposition not appearing in L or φ. The transformation that marks "valid" termination states is $\textsc{Term}(M, t) = (S, S_0, R', L', \Omega')$, where $R' = R \cup \{(s, s) \mid \forall s'.(s, s') \notin R\}$, $\Omega' = (p, q \vee t)$ and for a state s we set $L'(s) = L(s) \cup \{t\}$ if $\forall s'\ .\ (s, s') \notin R$ and $L'(s) = L(s)$ otherwise.

$$
\begin{aligned}
\textsc{Term}(\alpha, t) &::= \alpha \\
\textsc{Term}(\varphi_1 \wedge \varphi_2, t) &::= \textsc{Term}(\varphi_1, t) \wedge \textsc{Term}(\varphi_2, t) \\
\textsc{Term}(\varphi_1 \vee \varphi_2, t) &::= \textsc{Term}(\varphi_1, t) \vee \textsc{Term}(\varphi_2, t) \\
\textsc{Term}(\mathsf{EX}\varphi, t) &::= \neg t \wedge \mathsf{EX}(\textsc{Term}(\varphi, t)) \\
\textsc{Term}(\mathsf{AX}\varphi, t) &::= t \vee \mathsf{AX}(\textsc{Term}(\varphi, t)) \\
\textsc{Term}(\mathsf{EG}\varphi, t) &::= \mathsf{EG}\textsc{Term}(\varphi, t) \\
\textsc{Term}(\mathsf{AF}\varphi, t) &::= \mathsf{AF}\textsc{Term}(\varphi, t) \\
\textsc{Term}(\mathsf{A}[\varphi_1 \mathsf{W} \varphi_2], t) &::= \mathsf{A}[\textsc{Term}(\varphi_1, t) \ \mathsf{W}\ \textsc{Term}(\varphi_2, t)] \\
\textsc{Term}(\mathsf{E}[\varphi_1 \mathsf{U} \varphi_2], t) &::= \mathsf{E}[\textsc{Term}(\varphi_1, t) \ \mathsf{U}\ \textsc{Term}(\varphi_2, t)]
\end{aligned}
$$

Fig. 4. Transformation $\textsc{Term}(\varphi, t)$.

That is, we eliminate all finite paths in $\textsc{Term}(M, t)$ by instrumenting self loops and adding the proposition t on all terminal states. The fairness constraint is adjusted to include paths that end in such states. We now adjust the CTL formula φ that we wish to verify on M. Recall that t does not appear in φ. Now let $\textsc{Term}(\varphi, t)$ denote the CTL formula transformation in Fig. 4.

The combination of the two transformations maintains the validity of a CTL formula in a given system.

Theorem 1. $M \models_{\Omega_+} \varphi \Leftrightarrow \textsc{Term}(M, t) \models_{\Omega_+} \textsc{Term}(\varphi, t)$

Proof Sketch (full proof in [9]). For every fair path of $\textsc{Term}(M, t)$, we show that it corresponds to a maximal path in M and vice versa. The proof then proceeds by induction on the structure of the formula. For existential formulas, witnesses are translated between the models. For universal formulas, we consider arbitrary paths and translate them between the models. $\square$

After having marked the "valid" termination points in M by using the transformation $\textsc{Term}(M, t)$, we must ensure that our fair-CTL model-checking procedure ignores "invalid" finite paths in $\textsc{Fair}(M, \Omega)$. The finite paths that need to be removed from consideration are those that arise by wrong prediction of the prophecy variable n. The formula $\mathsf{term} = \mathsf{AFAX}\ \mathsf{false}$ holds in a state s iff all paths from s are finite. We denote its negation $\mathsf{EGEX}\ \mathsf{true}$ by $\neg\mathsf{term}$. Intuitively, when considering a state (s, n) of $\textsc{Fair}(M, \Omega)$, if (s, n) satisfies term, then (s, n) is part of a wrong prediction. If (s, n) satisfies $\neg\mathsf{term}$, then (s, n) is part of a correct prediction. Further on, we will set up our model checking technique such that universal path formulas ignore violations that occur on terminating paths (which correspond to wrong predictions) and existential path formulas use only non-terminating paths (which correspond to correct predictions).

3.3 Fair-CTL Model Checking

We use $\textsc{Fair}(M, \Omega)$ to handle fair-CTL model checking. Our procedure employs an existing CTL model checking algorithm for infinite-state systems. We assume that the CTL model checking algorithm returns an assertion characterizing all

```
let FairCTL(M, Ω, φ) : assertion =

    match(φ) with
    | Q φ1 OP φ2
    | φ1 bool_OP φ2 →
        a_φ1 = FairCTL(M, Ω, φ1);
        a_φ2 = FairCTL(M, Ω, φ2)
    | Q OP φ1 →
        a_φ1 = FairCTL(M, Ω, φ1)
    | α →
        a_φ1 = α

    match(φ) with
    | E φ1 U φ2 →
        φ' = E[a_φ1 U (a_φ2 ∧ ¬term)]
    | E Gφ1 →
        φ' = EG(a_φ1 ∧ ¬term)
    | E Xφ1 →
        φ' = EX(a_φ1 ∧ ¬term)
    | A φ1 W φ2 →
        φ' = A[a_φ1 W (a_φ2 ∨ term)]

    | A Fφ1 →
        φ' = AF(a_φ1 ∨ term)
    | A Xφ1 →
        φ' = AX(a_φ1 ∨ term)
    | φ1 bool_OP φ2 →
        φ' = a_φ1 bool_OP a_φ2
    | α →
        φ' = a_φ1

    M' = Fair(M, Ω)
    a = CTL(M', φ')

    match(φ) with
    | E φ' →
        return ∃n ≥ 0 . a
    | A φ' →
        return ∀n ≥ 0 . a
    | _ →
        return a
```

Fig. 5. Our procedure $\textsc{Fair}\mathrm{CTL}(M, \Omega, \varphi)$ which employs both an existing CTL model checker and the reduction $\textsc{Fair}(M, \Omega)$. An assertion characterizing the states in which φ holds under the fairness constraint Ω is returned.

```
let Verify(M, Ω, φ) : bool =

    a = FairCTL(Term(M, t), Ω, Term(φ, t))
    return S0 ⇒ a
```

Fig. 6. CTL model checking procedure $\textsc{Verify}$, which utilizes the subroutine in Fig. 5 to verify if a CTL property φ holds over M under the fairness constraints Ω.

the states in which a CTL formula holds. Tools proposed by Beyene *et al.* [2] and Cook *et al.* [10] support this functionality. We denote such CTL verification tools by $\mathrm{CTL}(M, \varphi)$, where M is a transition system and φ a CTL formula.

Our procedure adapting $\textsc{Fair}(M, \Omega)$ is presented in Fig. 5. Given a transition system M, a fairness constraint Ω, and a CTL formula φ, $\textsc{Fair}\mathrm{CTL}$ returns an assertion characterizing the states in which φ fairly holds. Initially, our procedure is called by $\textsc{Verify}$ in Fig. 6 where M and φ are initially transformed by $\textsc{Term}(M, t)$ and $\textsc{Term}(\varphi, t)$ discussed in Section 3.2. That is, $\textsc{Term}(M, t)$ marks all "valid" termination states in M to distinguish between maximal (finite) paths that occur in M and those introduced by our reduction. $\textsc{Term}(\varphi, t)$ allows us to disregard all aforementioned finite paths, as we only consider infinite paths, which correspond to a fair path in the original system.

Our procedure then begins by recursively enumerating over each CTL sub-property, wherein we attain an assertion characterizing all the states in which the sub-property holds under the fairness constraint Ω. These assertions will subsequently replace their corresponding CTL sub-properties as shown on lines 15,17,19, and so on. A new CTL formula φ' is then acquired by adding an appropriate termination or non-termination clause (lines 13-30). This clause allows us to ignore finite paths that are prefixes of unfair infinite paths. Recall that other finite paths were turned infinite and marked by the proposition t in $\textsc{Term}(M, t)$.

Ultimately, our reduction $\textsc{Fair}(M, \Omega)$ is utilized on line 32, where we transform the input transition system M according to Fig. 2. With our modified CTL formula φ' and transition system M', we call upon the existing CTL model checking algorithm to return an assertion characterizing all the states in which the formula holds. The returned assertion is then examined on lines 35-39 to determine whether or not φ' holds under the fairness constraint Ω. If the property is existential, then it is sufficient that there exists at least one value of the prophecy variable such that the property holds. If the property is universal, then the property must hold for all possible values of the prophecy variable.

We state the correctness and completeness of our model checking procedure.

Theorem 2. *For every CTL formula φ and every transition system M with no terminating states we have $M \models_{\Omega_+} \varphi \Leftrightarrow S_0 \to \textsc{FairCTL}\ (M, \Omega, \varphi)$.*

Proof Sketch (full proof in [9]). We show that every infinite path in $\textsc{Fair}(M, \Omega)$ starting in (s, n) for some $n \in \mathbb{N}$ corresponds to an infinite path in M starting in s satisfying Ω, and vice versa. From this correspondence of fair paths in M and infinite paths in $\textsc{Fair}(M, \Omega)$, we can safely disregard all the newly introduced finite paths given a transition system with no finite paths (i.e., $\textsc{Term}(M, t)$). $\square$

We then proceed to show by induction on the structure of the formula that the assertion returned by $\textsc{FairCTL}(M, \Omega, \varphi)$ characterizes the set of states of M that satisfy φ. For a universal property, we show that if it holds from s in M then it(s modified form) holds from (s, n) for every n in $\textsc{Fair}(M, \Omega)$ and vice versa. For an existential property, we show that if it holds from s in M then its modified form holds from (s, n) for some n in $\textsc{Fair}(M, \Omega)$ and vice versa.

Corollary 1. *For every CTL formula φ and every transition system M we have $M \models_{\Omega_+} \varphi \Leftrightarrow \textsc{Verify}(M, \Omega, \varphi)$ returns true.*

Proof. $\textsc{Verify}$ calls $\textsc{FairCTL}$ on $\textsc{Term}(M, t)$ and $\textsc{Term}(\varphi, t)$. It follows that $\textsc{Term}(M, t)$ has no terminating states and hence Theorem 2 applies to it. By Theorem 1, the mutual transformation of M to $\textsc{Term}(M, t)$ and φ to $\textsc{Term}(\varphi, t)$ preserves whether or not $M \models_{\Omega_+}$. The corollary follows. $\square$

4 Fair-ACTL Model Checking

In this section we show that in the case that we are only interested in universal path properties, i.e., formulas in ACTL, there is a simpler approach to fair-CTL model checking. In this simpler case, we can solely use the transformation

$$
\begin{aligned}
\text{NTERM}(\alpha) &::= \alpha \\
\text{NTERM}(\varphi_1 \wedge \varphi_2) &::= \text{NTERM}(\varphi_1) \wedge \text{NTERM}(\varphi_2) \\
\text{NTERM}(\varphi_1 \vee \varphi_2) &::= \text{NTERM}(\varphi_1) \vee \text{NTERM}(\varphi_2) \\
\text{NTERM}(\mathsf{AX}\varphi) &::= \mathsf{AX}(\text{NTERM}(\varphi) \vee \mathsf{term}) \\
\text{NTERM}(\mathsf{AF}\varphi) &::= \mathsf{AF}(\text{NTERM}(\varphi) \vee \mathsf{term}) \\
\text{NTERM}(\mathsf{A}[\varphi_1 \mathsf{W} \varphi_2]) &::= \mathsf{A}[\text{NTERM}(\varphi_1)\ \mathsf{W}\ (\text{NTERM}(\varphi_2) \vee \mathsf{term})]
\end{aligned}
$$

Fig. 7. Transformation NTERM( ).

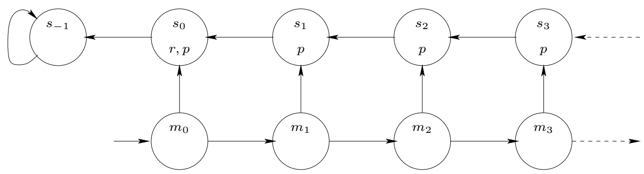


Fig. 8. A system showing that ECTL model checking is more complicated.

FAIR(M, Ω). Just like in FAIRCTL, we still must ignore truncated paths that correspond to wrong predictions. However, in this case, this can be done by a formula transformation.

Let NTERM(φ) denote the transformation in Figure 7. The transformation ensures that universal path quantification ignores states that lie on finite paths that are due to wrong estimations of the number of p-s until q. Using this transformation, it is possible to reduce fair-ACTL model checking to (A)CTL model checking over FAIR(M, Ω). Formally, this is stated in the following theorem.

Theorem 3. *For every ACTL formula φ and every transition system M with no terminating states, we have $M \models_{\Omega_+} \varphi \Leftrightarrow$ FAIR$(M, \Omega) \models$ NTERM$(\varphi) \vee$ term.*

Proof Sketch (full proof in [9]). The proof proceeds by induction on the structure of the formula. We show that if the property holds from s in M then for every n the (modified) property holds from (s, n) in FAIR(M, Ω) and vice versa. Note that the initial states of FAIR(M, Ω) are all the initial states of M annotated by all possible options of $n \in \mathbb{N}$. It follows that the combination of all transformations reduce fair ACTL model checking to ACTL model checking. □

Corollary 2. *For every ACTL formula φ we have*

$$M \models_{\Omega_+} \varphi \Leftrightarrow \text{FAIR}(\text{TERM}(M, t), \Omega) \models \text{NTERM}(\text{TERM}(\varphi, t)) \vee \textit{term}$$

Proof. As TERM(M, t) produces a transition system with no terminating states and TERM(φ, t) converts an ACTL formula to an ACTL formula, the proof then follows from Theorem 1 and Theorem 3. □

The direct reduction presented in Theorem 3 works well for ACTL but does not work for existential properties. We now demonstrate why Fig. 2 is not sufficient to handle existential properties alone. Consider the transition system M in Figure 8, the fairness constraint $\Omega = \{(p, q)\}$, and the property $\mathsf{EG}(\neg p \wedge \mathsf{EF}r)$. One can see that $M, m_0 \models_{\Omega_+} \mathsf{EG}(\neg p \wedge \mathsf{EF}r)$. Indeed, from each state s_i there is a unique path that eventually reaches s_0, where it satisfies r, and then continues to s_{-1}, where p does not hold. As the path visits finitely many p states it is clearly fair. So, every state m_i satisfies $\mathsf{EF}r$ by considering the path $m_i, s_i, s_{i-1}, \ldots, s_0, s_{-1}, \ldots$. Then the fair path $m_0, m_1, \ldots$ satisfies $\mathsf{EG}(\neg p \wedge \mathsf{EF}r)$. On the other hand, it is clear that no other path satisfies $\mathsf{EG}(\neg p \wedge \mathsf{EF}r)$.

Now consider the transformation $\textsc{Fair}(M, \Omega)$ and consider model checking of $\mathsf{EG}(\neg p \wedge \mathsf{EF}r)$. In $\textsc{Fair}(M, \Omega)$ there is no path that satisfies this property. To see this, consider the transition system $\textsc{Fair}(M, \Omega)$ and a value $n \in \mathbb{N}$. For every value of n the path $(m_0, n), (m_1, n), (m_2, n), \ldots$ is an infinite path in $\textsc{Fair}(M, \Omega)$ as it never visits p. This path does not satisfy $\mathsf{EG}(\neg p \wedge \mathsf{EF}r)$. Consider some state (m_j, n_j) reachable from (m_0, n) for $j > 2n$. The only infinite paths starting from (m_j, n_j) are paths that never visit the states s_i. Indeed, paths that visit s_i are terminated as they visit too many p states. Thus, for every $n \in \mathbb{N}$ we have $(m_0, n) \not\models \mathsf{EG}(\neg p \wedge \mathsf{EF}r)$. Finite paths in $\textsc{Fair}(M, \Omega)$ are those of the form $(m_0, n_0), \ldots, (m_i, n_i), (s_i, n_{i+1}), \ldots$. Such paths clearly cannot satisfy the property $\mathsf{EG}(\neg p \wedge \mathsf{EF}r)$ as the states s_i do satisfy p. Allowing existential paths to ignore fairness is clearly unsound. We note also that in $\textsc{Fair}(M, \Omega)$ we have $(m_0, n) \models \textsc{NTerm}(\mathsf{AF}(p \vee \mathsf{AG}\neg r))$.

Reducing Fair Termination to Termination. Given the importance of termination as a system property, we emphasize the reduction of fair termination to termination. Note that termination can be expressed in ACTL as $\mathsf{AFAX\ false}$, thus the results in Corollary 2 allow us to reduce fair termination to model checking (without fairness). Intuitively, a state that satisfies $\mathsf{AX\ false}$ is a state with no successors. Hence, every path that reaches a state with no successors is a finite path. Here, we demonstrate that for infinite-state infinite-branching systems, fair termination can be reduced to termination.

A transition system M terminates if for every initial state $s \in S_0$ we have $\Pi_\infty(s) = \emptyset$. System M fair-terminates under fairness Ω if for every initial state $s \in S_0$ and every $\pi \in \Pi_\infty(s)$ we have $\pi \not\models \Omega$, i.e., all infinite paths are unfair.

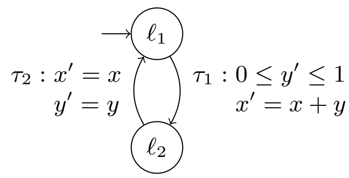
The following corollary follows from the proof of Theorem 3, where we establish a correspondence between fair paths of M and infinite paths of $\textsc{Fair}(M, \Omega)$.

Corollary 3. *M fair terminates iff $\textsc{Fair}(M, \Omega)$ terminates.*

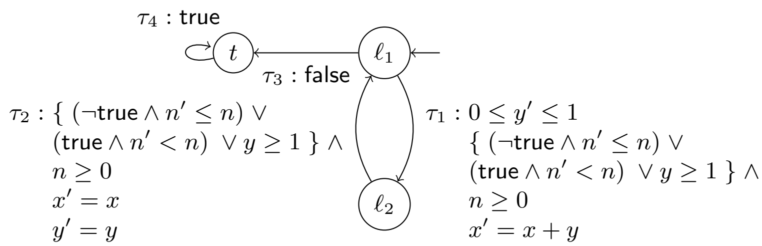
Recall that the reduction relies on transition systems having an infinite branching degree. For transition systems with finite-branching degree, we cannot reduce fair termination of finite-branching programs to termination of finite-branching programs, as the former is Σ_1^1-complete and the latter is RE-complete [18].

5 Example

Consider the example in Fig. 9. We will demonstrate the resulting transformations which will disprove the CTL property $\mathsf{EG}\ x \leq 0$ under the weak fairness



(a)



(b)

Fig. 9. Verifying a transition system with the CTL property $\mathsf{EG}\ x \leq 0$ and the weak fairness constraint $\mathsf{GF\ true} \rightarrow \mathsf{GF}\ y \geq 1$. The original transition system is represented in (a), followed by the application of our fairness reduction in (b).

constraint $\mathsf{GF\ true} \rightarrow \mathsf{GF\ y} \geq 1$ for the initial transition system introduced in (a). We begin by executing VERIFY in Fig. 6. In VERIFY the transition system in (a) is transformed according to TERM(M, t) and the CTL formula $\mathsf{EG}\ x \leq 0$ is transformed according to TERM(M, t), as discussed in 3.2. Our main procedure FAIRCTL in Fig. 5 is then called. First, we recursively enumerate over the most inner sub-property $x \leq 0$, wherein $x \leq 0$ is returned as it is our base case. In lines 13-30, a new CTL formula φ' is then acquired by adding an appropriate termination or non-termination clause. This clause allows us to ignore finite paths that are prefixes of some unfair infinite paths, that is, those that have not been marked by TERM(M, t). We then obtain (b) in Fig. 9 by applying FAIR(M, Ω) from Fig. 2 on line 32. Thus, we must restrict each transition such that $\{ \ (\neg\mathsf{true} \wedge n' \leq n) \ \vee (\mathsf{true} \wedge n' < n) \ \vee y \geq 1 \ \} \wedge n \geq 0$ holds. This can be seen in transitions τ_1 and τ_2.

Recall that FAIR(M, Ω) can include (maximal) finite paths that are prefixes of unfair infinite paths. We thus have to ensure that these paths do not interfere with the validity of our model checking procedure. We have shown how to distinguish between maximal (finite) paths that occur in M and those introduced by our transformation in Theorem 1. This is demonstrated by τ_3 and τ_4 in (b): in τ_3 we simply take the negation of the loop invariant (in this case it is false), as it would indicate a terminating path given that no other transitions follow the loop termination. In τ_4 we instrument a self loop and add the proposition t to eliminate all terminal states. Additionally, utilizing TERM(φ, t) on $\mathsf{EG}\ x \leq 0$ allows us to disregard all aforementioned marked finite paths, as we only consider infinite paths which correspond to a fair path in the original system.

On line 33, a CTL model checker is then employed with the transition system in (b) and the CTL formula φ'. We then apply tools provided by Beyene *et al.* [2] and Cook *et al.* [10]to the transformation introduced to verify CTL for infinite-state systems. An assertion characterizing the states in which φ' holds is returned and then further examined on lines 36 and 37, where it is discovered that this property does not hold due to the restrictive fairness constraint applied to the existential CTL property. The weak fairness constraint requires that infinitely often $y \geq 1$ holds, which interferes with the existential property that $\mathsf{EG}\ x \leq 0$. This shows that for the existential fragment of CTL, fairness constraints restrict the transition relations required to prove an existential property. This can be beneficial when attempting to disprove systems and their negations.

6 Experiments

In this section we demonstrate the results of preliminary experiments with a prototype implementation. We applied our tool to several small programs: a classical mutual exclusion algorithm as well as code fragments drawn from device drivers. Our implementation is based on an extension to T2 [4], [14], [10].[6] As previously discussed, there are currently no known tools supporting fair-CTL for infinite-state systems, thus we are not able to make experimental comparisons.

Fig. 10 shows experimental evaluations of sequential Windows device drivers (WDD) and various concurrent systems [7]. WDD1 uses the fairness constraint $\mathsf{GF}(\texttt{IoCreateDevice.exit}\{1\}) \Rightarrow \mathsf{GF}(\texttt{status} = \texttt{SUCCESS})$, while WDD2 and 3 utilize the same fairness constraint in relation to checking the acquisition and release of spin locks and the entrance and exit of critical regions, respectively. WDD4 requires a weak fairness constraint indicating that `STATUS_OK` will hold a value of `true` infinitely often, that is, whenever sockets are successfully opened, the server will eventually return a successful status infinitely often.

Note that the initially concurrent programs are reduced to sequential programs via [7], which uses rely-guarantee reasoning to reduce multi-threaded verification to liveness. We verify the traditional Bakery algorithm, requiring that any thread requesting access to the critical region will eventually be granted the right to do so. The producer-consumer algorithm requires that any amount of input data produced, must be eventually consumed. The Chain benchmark consists of a chain of threads, where each thread decreases its own counter, but the next thread in the chain can counteract, and increase the counter of the previous thread, thus only the last thread in the chain can be be decremented unconditionally. These algorithms are verified on 2, 4, and 8 threads, respectively.

For the the existential fragment of CTL, fairness constraints can often restrict the transition relations required to prove an existential property, as demonstrated by WDD3. For universal CTL properties, fairness policies can assist in

[6] T2 can be acquired at http://research.microsoft.com/en-us/projects/t2/

[7] Benchmarks can be found at http://heidyk.com/experiments.html

Program	LOC	Property	FC	Time(s)	Result
WDD1	20	$\mathsf{AG}(\mathtt{BlockInits()} \Rightarrow \mathsf{AF}\ \mathtt{UnblockInits()})$	Yes	14.4	✓
WDD1	20	$\mathsf{AG}(\mathtt{BlockInits()} \Rightarrow \mathsf{AF}\ \mathtt{UnblockInits()})$	No	2.1	χ
WDD2	374	$\mathsf{AG}(\mathtt{AcqSpinLock()} \Rightarrow \mathsf{AF}\ \mathtt{RelSpinLock()})$	Yes	18.8	✓
WDD2	374	$\mathsf{AG}(\mathtt{AcqSpinLock()} \Rightarrow \mathsf{AF}\ \mathtt{RelSpinLock()})$	No	14.1	χ
WDD3	58	$\mathsf{AF}(\mathtt{EnCritRegion()} \Rightarrow \mathsf{EG}\ \mathtt{ExCritRegion()})$	Yes	12.5	χ
WDD3	58	$\mathsf{AF}(\mathtt{EnCritRegion()} \Rightarrow \mathsf{EG}\ \mathtt{ExCritRegion()})$	No	9.6	✓
WDD4	302	$\mathsf{AG}(\mathtt{added\_socket} > 0 \Rightarrow \mathsf{AFEG}\ \mathtt{STATUS\_OK})$	Yes	30.2	✓
WDD4	302	$\mathsf{AG}(\mathtt{added\_socket} > 0 \Rightarrow \mathsf{AFEG}\ \mathtt{STATUS\_OK})$	No	72.4	χ
Bakery	37	$\mathsf{AG}(\mathtt{Noncritical} \Rightarrow \mathsf{AF}\ \mathtt{Critical})$	Yes	2.9	✓
Bakery	37	$\mathsf{AG}(\mathtt{Noncritical} \Rightarrow \mathsf{AF}\ \mathtt{Critical})$	No	16.4	χ
Prod-Cons	30	$\mathsf{AG}(\mathtt{p_i} > 0 \Rightarrow \mathsf{AF}\ \mathtt{q_i} <= 0)$	Yes	18.5	✓
Prod-Cons	30	$\mathsf{AG}(\mathtt{p_i} > 0 \Rightarrow \mathsf{AF}\ \mathtt{q_i} <= 0)$	No	5.5	χ
Chain	48	$\mathsf{AG}(\mathtt{x} \geq 8 \Rightarrow \mathsf{AF}\ \mathtt{x} = 0)$	Yes	1.8	✓
Chain	48	$\mathsf{AG}(\mathtt{x} \geq 8 \Rightarrow \mathsf{AF}\ \mathtt{x} = 0)$	No	4.7	χ

Fig. 10. Experimental evaluations of infinite-state programs such as Windows device drivers (WDD) and concurrent systems, which were reduced to non-deterministic sequential programs via [7]. Each program is tested for both the success of a branching-time liveness property with a fairness constraint and its failure due to a lack of fairness. A ✓ represents the existence of a validity proof, while χ represents the existence of a counterexample. We denote the lines of code in our program by LOC and the fairness constraint by FC. There exist no competing tools available for comparison.

enforcing properties to hold that previously did not. Thus, our tool allows us to both prove and disprove the negation of each of the properties.

7 Discussion

We have described the first-known fair-CTL model checking technique for integer based infinite-state programs. Our approach is based on a reduction to existing techniques for fairness-free CTL model checking. The reduction relies on utilizing prophecy variables to introduce additional information into the state-space of the program under consideration. This allows fairness-free CTL proving techniques to reason only about fair executions. Our implementation seamlessly builds upon existing CTL proving techniques, resulting in experiments which demonstrate the practical viability of our approach.

Furthermore, our technique allows us to bridge between linear-time (LTL) and branching-time (CTL) reasoning. Not only so, but a seamless integration between LTL and CTL reasoning may make way for further extensions supporting CTL* verification of infinite-state programs [16]. We hope to further examine both the viability and practicality of such an extension.

We include the definition of fair-CTL considering only infinite paths and show how to change transition systems to use either definition in our technical report which can be acquired at [9]. Additionally, we show how to modify the proof system to incorporate an alternative approach to CTL verification advocated by Cook & Koskinen [12].

References

1. K. Apt and E. Olderog. Fairness in parallel programs: The transformational approach. *ACM TOPLAS*, 10, 1988.
2. T. Beyene, C. Popeea, and A. Rybalchenko. Solving existentially quantified horn clauses. In *CAV*, LNCS, 2013.
3. N. Bjørner, A. Browne, M. Colón, B. Finkbeiner, Z. Manna, H. Sipma, and T. Uribe. Verifying temporal properties of reactive systems: A step tutorial. *FMSD*, 16(3), 2000.
4. M. Brockschmidt, B. Cook, and C. Fuhs. Better termination proving through cooperation. In *CAV*, LNCS, 2013.
5. A. Cimatti, E.M. Clarke, G. Enrico, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *CAV*, LNCS, 2002.
6. E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM TOPLAS*, 8(2), 1986.
7. B. Cook, A. Gotsman, M. Parkinson, and V. Vafeiadis. Proving that non-blocking algorithms don't block. In *POPL*. ACM, 2009.
8. B. Cook, A. Gotsman, A. Podelski, A. Rybalchenko, and M. Vardi. Proving that programs eventually do something good. In *POPL*. ACM, 2007.
9. B. Cook, Khlaaf H, and N. Piterman. Fairness for infinite-state systems. TR RN/14/11, UCL, 2014.
10. B. Cook, H. Khlaaf, and N. Piterman. Faster temporal reasoning for infinite-state programs. In *FMCAD*, LNCS, 2014.
11. B. Cook and E. Koskinen. Making prophecies with decision predicates. In *POPL*. ACM, 2011.
12. B. Cook and E. Koskinen. Reasoning about nondeterminism in programs. In *PLDI*. ACM, 2013.
13. B. Cook, E. Koskinen, and M. Vardi. Temporal property verification as a program analysis task. In *CAV*, LNCS, 2011.
14. B. Cook, A. See, and F. Zuleger. Ramsey vs. lexicographic termination proving. In *TACAS*, LNCS, 2013.
15. A. David, J. Håkansson, K.G. Larsen, and P. Pettersson. Model checking timed automata with priorities using dbm subtraction. In *FORMATS*, 2006.
16. E.A. Emerson and J.Y Halpern. "sometimes" and "not never" revisited: On branching versus linear time temporal logic. *J. ACM*, 33(1), January 1986.
17. E.A. Emerson and C.-L. Lei. Temporal reasoning under generalized fairness constraints. In *STACS*, LNCS, 1986.
18. D. Harel. Effective transformations on infinite trees, with applications to high undecidability, dominoes and fairness. *J. ACM*, 33:224–248, 1986.
19. M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *CAV*, LNCS, 2011.
20. A. Pnueli, A. Podelski, and A. Rybalchenko. Separating fairness and well-foundedness for the analysis of fair discrete systems. In *TACAS*, LNCS, 2005.
21. A. Pnueli and Y. Sa'ar. All you need is compassion. In *VMCAI*, LNCS, 2008.
22. A. Podelski and A. Rybalchenko. Transition invariants. In *LICS*, 2004.
23. S. Schwoon. Moped - A Model-Checker for Pushdown Systems. http://www7.in.tum.de/ schwoon/moped, 2002.
24. F. Song and T. Touili. Pushdown model checking for malware detection. In *ESEC/FSE*, 2013.
25. M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *I&C*, 115(1):1–37, 1994.