

Abstract: Fairness for Infinite-State Systems

Byron Cook^{1,2}, Heidy Khlaaf², and Nir Piterman³

1 Microsoft Research

2 University College London

3 University of Leicester

Abstract

In this extended abstract 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. The key idea is to use prophecy variables in the reduction for the purpose of symbolically partitioning fair from unfair executions.

1998 ACM Subject Classification D.2.4 Software/Program Verification, F.3.1 Specifying and Verifying and Reasoning about Programs, I.2.2 Automatic Programming

Keywords and phrases Model checking, Temporal logic, Fair CTL, Fair termination

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 [11]. Furthermore, when proving state-based CTL properties, we must often use fairness to model trace-based assumptions about the environment.

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. We use the prophecy to encode a partition of fair from unfair paths. Prophecy variables introduce additional information into the state-space of the program under consideration, thus allowing fairness-free CTL proving techniques to reason only about fair executions.

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 [10].¹ 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: Recent approaches (*e.g.* [2, 6, 7]) facilitate model-checking fairness-free CTL over programs with infinite (discrete) non-deterministic branching. We use these results to provide a model checking procedure for fair CTL. As a consequence, in the context of infinite branching, fair and fairness-free CTL are equally difficult (and similarly for termination).

With our new technique we can build practical tools for automatically proving fair CTL of programs. We show the viability of our approach in practice using examples drawn from device drivers and algorithms utilizing shared resources.

¹ Sometimes termination refers to *universal termination*, which entails termination for *all* possible inputs. This is a harder problem and is co-RE^{RE}-complete.

$$\begin{aligned}
& \text{FAIR}((S, S_0, R, L), (p, q)) \triangleq (S_\Omega, S_\Omega^0, R_\Omega, L_\Omega) \\
& \text{where} \\
& S_\Omega = S \times \mathbb{N} \\
& R_\Omega = \{((s, n), (s', n')) \mid (s, s') \in R\} \wedge \begin{pmatrix} (\neg p \wedge n' \leq n) \vee \\ (p \wedge n' < n) \vee \\ q \end{pmatrix} \\
& S_\Omega^0 = S_0 \times \mathbb{N} \\
& L_\Omega(s, n) = L(s)
\end{aligned}$$

■ **Figure 1** 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 $(S_\Omega, S_\Omega^0, R_\Omega, L_\Omega)$. Note that $n \geq 0$ is implicit, as $n \in \mathbb{N}$.

1.1 Intuition

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 an illustrative example adapted for using said transformation, and subsequently our experimental results.

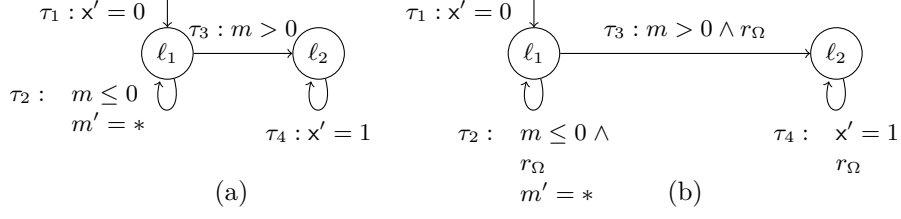
In Fig. 1, we propose a reduction $\text{FAIR}(M, \Omega)$ that encodes an instantiation of the fairness constraint within a transition system. A transition system is $M = (S, S_0, R, L)$, where S is a countable set of states, $S_0 \subseteq S$ a set of initial states, $R \subseteq S \times S$ a transition relation, and $L : S \rightarrow 2^{AP}$ 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. When given a transition system (S, S_0, R, L) and a strong fairness constraint $\Omega = (p, q)$ where $p, q \subseteq S$, $\text{FAIR}(M, \Omega)$ returns a new transition system 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.

The conversion of M with fairness constraint Ω to $\text{FAIR}(M, \Omega)$ involves the truncation of paths due to the wrong estimation of the number of p -s until q . This means that $\text{FAIR}(M, \Omega)$ can include (maximal) finite paths that are prefixes of unfair infinite paths. It follows that when model checking CTL we have to ensure that these paths do not interfere with the validity of our model checking procedure. Hence, we have to distinguish between maximal (finite) paths that occur in M and those introduced by our reduction. This is done through adding a proposition t to mark all original “valid” termination states prior to the reduction in Fig. 1, followed by adjusting the CTL specification through a transformation.

2 Illustrative Example

We first provide high-level understanding of our approach through an example.

Consider the example in Fig. 2 for the CTL property $\text{AG}(x = 0 \rightarrow \text{AF}(x = 1))$ and the fairness constraint $\text{GF } \tau_2 \rightarrow \text{GF } m > 0$ for the initial transition system introduced in (a). That is, we are attempting to prove that for all states, when $x = 0$ then we must always eventually reach a state such that $x = 1$ under the fairness constraint that if the transition τ_2 occurs infinitely often, then m must be greater than 0 infinitely often. We demonstrate the resulting transformation for this infinite-state program which allows us to reduce fair model checking to model checking. By applying $\text{FAIR}(M, \Omega)$ from Fig. 1, we obtain (b) where each original transition, τ_2, τ_3 , and τ_4 , are adjoined with restrictions such



$$r_\Omega : \{ (\neg\tau_2 \wedge n' \leq n) \vee (\tau_2 \wedge n' < n) \vee m > 0 \} \wedge n \geq 0$$

■ **Figure 2** Reducing a transition system with the CTL property $\text{AG}(x = 0 \rightarrow \text{AF}(x = 1))$ and the weak fairness constraint $\text{GF } \tau_2 \rightarrow \text{GF } m > 0$. The original transition system is represented in (a), followed by the application of our fairness reduction in (b).

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 > 0 infinitely often. In τ_2 , the assignment $m' = *$ 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 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, 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. The transformed figure in (b) can then be employed by an existing CTL model checking algorithm for infinite-state systems in order to verify the input CTL formula. We assume that the CTL model checking algorithm returns an assertion characterizing all the states in which a CTL formula holds. Tools such as Beyene *et al.* [2] and Cook *et al.* [4] support this functionality.

3 Experiments

We discuss 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 [3, 8].² Despite theoretical contributions to the topic of fair CTL for infinite-state programs [1], there are no known tools supporting fair CTL for infinite-state programs. We are thus unable to make experimental comparisons.

Fig. 3 shows the results of our experiments. In our experiments we verified liveness properties, expressed in CTL. For each program we tested for both the success of the liveness property with a fairness constraint and its failure due to either a lack of fairness or a presence of a bug. A $\checkmark$ 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.

Note that the Bakery algorithm is meant to be performed on a multi-threaded program. Due to our lack of support for concurrency, we have re-written the algorithm sequentially in

² New versions of T2 are not publicly available due to legal constraints. However, we are currently working through a release process with the Microsoft legal team.

Program	LOC	Property	FC	Time(s)	Result
Windows Device Driver 1	20	$AG(PBBlockInits() \Rightarrow AFPPUnblockInits())$	Yes.	14.43	✓
Windows Device Driver 1	20	$AG(PBBlockInits() \Rightarrow AFPPUnblockInits())$	No.	2.16	χ
Windows Device Driver 1 + bug	20	$AG(PBBlockInits() \Rightarrow AFPPUnblockInits())$	Yes.	10.11	χ
Bakery	37	$AG(Noncritical \Rightarrow AFCritical)$	No.	16.43	✓
Bakery	37	$AG(Noncritical \Rightarrow AFCritical)$	Yes.	2.98	χ
Bakery + bug	37	$AG(Noncritical \Rightarrow AFCritical)$	No.	12.48	χ
Windows Device Driver 2	374	$AG(KeAcquireSpinLock() \Rightarrow AFKeReleaseSpinLock())$	Yes.	18.84	✓
Windows Device Driver 2	374	$AG(KeAcquireSpinLock() \Rightarrow AFKeReleaseSpinLock())$	No.	14.12	χ
Windows Device Driver 2 + bug	374	$AG(KeAcquireSpinLock() \Rightarrow AFKeReleaseSpinLock())$	Yes.	18.94	χ
Windows Device Driver 3	58	$AF(KeEnCriticalRegion() \Rightarrow EGKeExCriticalRegion())$	Yes.	12.58	χ
Windows Device Driver 3	58	$AF(KeEnCriticalRegion() \Rightarrow EGKeExCriticalRegion())$	No.	9.62	✓

■ **Figure 3** Windows Device Driver 1 uses the fairness constraint $GF(IoCreateDevice.exit\{1\}) \Rightarrow GF(status! = STATUS_OBJ_NAME_COLLISION)$. Windows Device Driver 2 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. The Bakery algorithm utilizes a fairness constraint of the form $GF(p) \Rightarrow GF(q)$ with p and q being relative to our sequential implementation.

a manner which simulates the behavior of a multi-threaded program. The variables p and q in the fairness constraint of Bakery denote specific program locations in our sequential algorithm.

For the existential fragment of CTL, fairness constraints restrict the transition relations required to prove an existential property, as demonstrated by Windows Device Driver 3. For universal CTL properties, fairness policies can assist in 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.

4 Discussion

We have shown the first-known fair-CTL model checking technique for integer based infinite-state programs through 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. Cook *et al.* [5] have described an iterative symbolic determination procedure which efficiently uses techniques associated with CTL to verify a subset of LTL properties. However, these corresponding branching time techniques provide no support for verifying fair-CTL, thus excluding a large set of linear-time liveness properties necessitating fairness. Our contribution would thus allow for full support of LTL verification via CTL model checking techniques. 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 [9]. We hope to further examine both the viability and practicality of such an extension.

References

- 1 K. Apt and E. Olderog. Fairness in parallel programs: The transformational approach. *ACM Transactions on Programming Languages and Systems*, 10, 1988.
- 2 T. Beyene, C. Popeea, and A. Rybalchenko. Solving existentially quantified horn clauses. In *Proceedings of the 25th International Conference on Computer Aided Verification, CAV'13*, 2013.
- 3 M. Brockschmidt, B. Cook, and C. Fuhs. Better termination proving through cooperation. In *Proceedings of the 25th International Conference on Computer Aided Verification, CAV'13*, 2013.
- 4 B. Cook, H. Khlaaf, and N. Piterman. Faster temporal reasoning for infinite-state programs. Technical Report CS-14-001, University of Leicester, 2014.
- 5 B. Cook and E. Koskinen. Making prophecies with decision predicates. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '11*, 2011.
- 6 B. Cook and E. Koskinen. Reasoning about nondeterminism in programs. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 219–230. ACM, 2013.
- 7 B. Cook, E. Koskinen, and M. Vardi. Temporal property verification as a program analysis task. In *Proceedings of the 23rd International Conference on Computer Aided Verification, CAV'11*, 2011.
- 8 B. Cook, A. See, and F. Zuleger. Ramsey vs. lexicographic termination proving. In *Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'13*, 2013.
- 9 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.
- 10 D. Harel. Effective transformations on infinite trees, with applications to high undecidability, dominoes and fairness. *Journal of the ACM*, 33:224–248, 1986.
- 11 M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Inf. Comput.*, 115(1):1–37, 1994.