Information and Computation **173**, 64–81 (2002)
doi:10.1006/inco.2001.3085, available online at http://www.idealibrary.com on IDEAL®

Fair Simulation[1]

Thomas A. Henzinger

EECS Department, University of California, Berkeley, California 94720-1770
E-mail: tah@eecs.berkeley.edu

Orna Kupferman

School of Computer Science and Engineering, Hebrew University, Jerusalem 91904, Israel
E-mail: orna@cs.huji.ac.il

and

Sriram K. Rajamani

Microsoft Research, One Microsoft Way, Redmond, Washington 98052
E-mail: sriram@microsoft.com

Received February 1, 1998

The simulation preorder for labeled transition systems is defined locally, and operationally, as a game that relates states with their immediate successor states. Simulation enjoys many appealing properties. First, simulation has a denotational characterization: system S simulates system I iff every computation tree embedded in the unrolling of I can be embedded also in the unrolling of S. Second, simulation has a logical characterization: S simulates I iff every universal branching-time formula satisfied by S is satisfied also by I. It follows that simulation is a suitable notion of implementation, and it is the coarsest abstraction of a system that preserves universal branching-time properties. Third, based on its local definition, simulation between finite-state systems can be checked in polynomial time. Finally, simulation implies trace containment, which cannot be defined locally and requires polynomial space for verification. Hence simulation is widely used both in manual and in automatic verification. Liveness assumptions about transition systems are typically modeled using fairness constraints. Existing notions of simulation for fair transition systems, however, are not local, and as a result, many appealing properties of the simulation preorder are lost. We propose a new view of fair simulation by extending the local definition of simulation to account for fairness: system $\mathcal{S}$ *fairly simulates* system $\mathcal{I}$ iff in the simulation game, there is a strategy that matches with each fair computation of $\mathcal{I}$ a fair computation of $\mathcal{S}$. Our definition enjoys a denotational characterization and has a logical characterization: $\mathcal{S}$ fairly simulates $\mathcal{I}$ iff every fair computation tree (whose infinite paths are fair) embedded in the unrolling of $\mathcal{I}$ can be embedded also in the unrolling of $\mathcal{S}$ or, equivalently, iff every Fair-∀AFMC formula satisfied by $\mathcal{S}$ is satisfied also by $\mathcal{I}$ (∀AFMC is the universal fragment of the alternation-free μ-calculus). The locality of the definition leads us to a polynomial-time algorithm for checking fair simulation for finite-state systems with weak and strong fairness constraints. Finally, fair simulation implies fair trace containment and is therefore useful as an efficiently computable local criterion for proving linear-time abstraction hierarchies of fair systems. © 2002 Elsevier Science (USA)

Key Words: verification; simulation; fairness.

1. INTRODUCTION

In program verification, we check that an implementation satisfies a specification. Both the implementation and the specification describe the possible behaviors of a program at different levels of abstraction. We distinguish between two approaches to satisfaction of a specification by an implementation. In *trace-based* satisfaction, we require that every linear property (i.e., every property of

[1] A preliminary version of this paper appeared in *CONCUR 97: Concurrency Theory*, Springer-Verlag, Lecture Notes in Computer Science, Vol. 1243, pp. 273–287, 1997. The research was supported in part by ONR YIP Award N00014-95-1-0520, by NSF CAREER Award CCR-9501708, by NSF Grant CCR-9504469, by ARPA Grant NAG2-892, and by SRC Contract 95-DC-324.036.

computation sequences) which holds for the specification holds also for the implementation. In *tree-based* satisfaction, we require that every branching property (i.e., every property of computation trees) which holds for the specification holds also for the implementation [26].

If we represent the implementation $\mathcal{I}$ and the specification $\mathcal{S}$ using state-transition systems, then the formal relation that captures trace-based satisfaction is trace containment: $\mathcal{S}$ *trace-contains* $\mathcal{I}$ iff it is possible to generate by $\mathcal{S}$ every (finite and infinite) sequence of observations that can be generated by $\mathcal{I}$. The notion of trace containment is robust with respect to linear temporal logics such as LTL in the sense that $\mathcal{S}$ trace-contains $\mathcal{I}$ iff every LTL formula that holds for $\mathcal{S}$ holds also for $\mathcal{I}$. Unfortunately, it is difficult to check trace containment (complete for PSPACE [31]), and we are unlikely to find an efficient algorithm.

The formal relation that captures tree-based satisfaction is tree containment: $\mathcal{S}$ *tree-contains* $\mathcal{I}$ iff it is possible to embed in the unrolling of $\mathcal{S}$ every (finite and infinite) tree of observations that can be embedded in the unrolling of $\mathcal{I}$. The notion of tree containment is equivalent to the notion of *simulation*, as defined by Milner [25]: $\mathcal{S}$ tree-contains $\mathcal{I}$ iff $\mathcal{S}$ simulates $\mathcal{I}$; that is, we can relate each state of $\mathcal{I}$ to a state of $\mathcal{S}$ so that two related states i and s agree on their observations and every successor of i is related to some successor of s.

Simulation has several theoretically and practically appealing properties. First, like trace containment, simulation is robust: for universal branching temporal logics (where only universal path quantification is allowed) such as ∀CTL (the universal fragment of computation tree logic), ∀CTL*, and ∀AFMC (the universal fragment of the alternation free μ-calculus), $\mathcal{S}$ simulates $\mathcal{I}$ iff every formula that holds for $\mathcal{S}$ holds also for $\mathcal{I}$ [3, 11]. Second, unlike trace containment, the definition of simulation is local, as the relation between two states is based only on their successor states. As a result, it can be checked in polynomial time (quadratic in both $\mathcal{S}$ and $\mathcal{I}$) whether $\mathcal{S}$ simulates $\mathcal{I}$ [5, 13], and a witnessing relation for simulation can be computed using a symbolic fixpoint procedure [13]. The locality advantage is so compelling as to make simulation useful also to researchers that favor trace-based specification: in automatic verification, simulation is widely used as an efficiently computable sufficient condition for trace containment [8, 16]; in manual verification, trace containment is most naturally proved by exhibiting local witnesses such as simulation relations or refinement mappings (a restricted form of simulation relations) [9, 21, 23, 24].[2]

State-transition systems describe only the *safe* behaviors of programs. In order to model *liveness* assumptions, one typically augments state-transition systems with fairness constraints, which partition the infinite computations of a system into fair and unfair computations. The linear framework of trace containment generalizes naturally to fair trace containment: $\mathcal{S}$ *fairly trace-contains* $\mathcal{I}$ iff it is possible to fairly generate by $\mathcal{S}$ every sequence of observations that can be fairly generated by $\mathcal{I}$. Robustness with respect to LTL and PSPACE-completeness extend to the fair case.

It is not so obvious how to generalize the branching framework of simulation to account for fairness. Indeed, several proposals can be found in the literature. The definition suggested by Grumberg and Long [11] rests on the motivation that $\mathcal{S}$ fairly simulates $\mathcal{I}$ iff every Fair-∀CTL* formula that holds for $\mathcal{S}$ holds also for $\mathcal{I}$ (the universal path quantifier of Fair-∀CTL* ranges over fair computations only). This definition, however, is neither robust (Fair-∀CTL induces a weaker preorder [2], and Fair-∀AFMC, as we show here, induces a stronger one) nor can it be checked efficiently (it is complete for PSPACE [18]). Following [16], we call the Grumberg–Long version of fair simulation ∃-*simulation*, because it can be defined as simulation where each fair computation of $\mathcal{I}$ is related to *some* fair computation of $\mathcal{S}$. In manual verification, by Lynch and others [23, 24], usually a stronger notion of fair simulation is used, which we call ∀-*simulation* (see also [10]): for each fair computation of $\mathcal{I}$, *every* related computation of $\mathcal{S}$ is required to be fair.[3] Again, this definition is neither robust (no logical characterization is known) nor can it be checked efficiently (it is NP-complete [16]). While both ∃-simulation and ∀-simulation are sufficient conditions for fair trace containment, they do not provide any computational advantage (indeed, algorithms for checking ∃-simulation use subroutines for checking fair trace containment).

We introduce a new definition of *fair simulation* and argue for its theoretical and practical merits and its advantages over existing definitions. In order to define fair simulation without losing the locality that

[2] In [1], it is shown that if auxiliary observable variables may be added to a system, then simulation is not only a sound proof technique but also complete for proving trace containment.

[3] Using a similar proof technique, Lamport and others [1, 21] suggest a restricted, functional version of simulation, called *refinement mapping*. There, every computation of $\mathcal{I}$ is related to exactly one computation of $\mathcal{S}$; thus, ∃-simulation coincides with ∀-simulation.

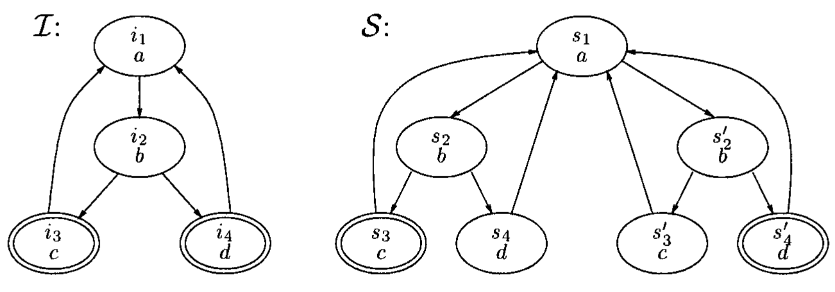


FIG. 1. Fair simulation is stronger than ∃-simulation.

makes simulation useful, we go back to the basis of the branching-time approach and view simulation as a generalization of tree containment to *fair tree containment*: we define that $\mathcal{S}$ *fairly simulates* $\mathcal{I}$ iff it is possible to fairly embed in the unrolling of $\mathcal{S}$ every tree of observations that can be fairly embedded in the unrolling of $\mathcal{I}$, where a tree embedding is fair if all infinite paths are mapped onto fair computations.[4] This definition falls strictly between ∃-simulation and ∀-simulation.

Consider the implementation and specification appearing in Fig. 1. Each state is labeled with an observation from the set $\{a, b, c, d\}$. The structures representing them are augmented with Büchi fairness constraints. In order to be fair, an infinite computation of $\mathcal{I}$ must visit the set $\{i_3, i_4\}$ infinitely often, and an infinite computation of $\mathcal{S}$ must visit the set $\{s_3, s'_4\}$ infinitely often. It is easy to see that $\mathcal{S}$ ∃-simulates $\mathcal{I}$. Indeed, the relation that maps each state in $\mathcal{I}$ to the set of states in $\mathcal{S}$ that agree with its observation is an ∃-simulation. For example, the fair computation $(i_1, i_2, i_3, i_1, i_2, i_4)^\omega$ of $\mathcal{I}$ is related to the fair computation $(s_1, s_2, s_3, s_1, s'_2, s'_4)^\omega$ of $\mathcal{S}$. Nevertheless, the infinite tree generated by unwinding $\mathcal{I}$ cannot be fairly embedded in an unwinding of $\mathcal{S}$. To see this, note that every occurrence of s_2 in any embedding must have both s_3 and s_4 as successors. Similarly, every occurrence of s'_2 must have both s'_3 and s'_4 as successors. Consequently, any embedding must have an infinite unfair computation.

Consider now the implementation and specification appearing in Fig. 2. Here, the infinite fair computations of $\mathcal{I}$ are those that visit i_6 infinitely often, and the infinite fair computations of $\mathcal{S}$ are those that visit s_6 infinitely often. Clearly, we can fairly embed in $\mathcal{S}$ the tree generated by unwinding $\mathcal{I}$. Hence, $\mathcal{S}$ fairly simulates $\mathcal{I}$. Still, $\mathcal{S}$ does not ∀-simulate $\mathcal{I}$. To see this, note that any candidate relation for ∀-simulation must relate i_6 to both s_4 and s_5. Then, however, the observations along the unfair computation $s_1 \cdot s_2 \cdot (s_4 \cdot s_5)^\omega$ of $\mathcal{S}$ agree with the observations of the fair computation $i_1 \cdot i_2 \cdot i_6^\omega$ of $\mathcal{I}$.

The definition of fair simulation as fair tree containment is equivalent to an alternative, local definition that is based on games. It is well known that $\mathcal{S}$ simulates $\mathcal{I}$ iff in an infinite game of the protagonist $\mathcal{S}$ against the antagonist $\mathcal{I}$, the protagonist can match every move of the antagonist by moving to a state with the same observation. Then, $\mathcal{S}$ fairly simulates $\mathcal{I}$ iff the protagonist has a strategy such that in the limit, after ω moves, if the antagonist produces a fair computation of $\mathcal{I}$, then the protagonist produces a fair computation of $\mathcal{S}$. Consider again implementation $\mathcal{I}$ and specification $\mathcal{S}$ of Fig. 1, with the antagonist starting at i_1 and the protagonist starting at s_1. We show that the antagonist can produce a fair computation of $\mathcal{I}$ that the protagonist cannot match. The antagonist first moves to i_2 and then uses the following strategy. If the protagonist replies with a move to s_2, then the antagonist makes its next move to i_4, forcing the protagonist to reply with a move to s_4. If, on the other hand, the protagonist replies with a move to s'_2, then the antagonist makes its next move to i_3, forcing the protagonist to reply with a move to s'_3. Keeping to this strategy, the antagonist produces a fair computation (all infinite computations of $\mathcal{I}$ are fair), while the protagonist, irrespective of the strategy used, produces an unfair computation, which never visits s_3 or s'_4. Hence, $\mathcal{S}$ does not fairly simulate $\mathcal{I}$. By contrast, recall $\mathcal{S}$ ∃-simulates $\mathcal{I}$: while in ∃-simulation the protagonist can make use of information about the future moves of the antagonist in order to produce a fair computation, in fair simulation the strategy of the protagonist must depend on the past and current moves of the antagonist only.

We argue that our definition of fair simulation is a suitable extension of simulation to fairness, as it preserves many of the appealing properties of simulation:

[4] Note that while in the nonfair case, it suffices to embed the unrolling of $\mathcal{I}$ in the unrolling of $\mathcal{S}$, the unrolling of $\mathcal{I}$ may involve unfair paths. This necessitates the more involved definition.

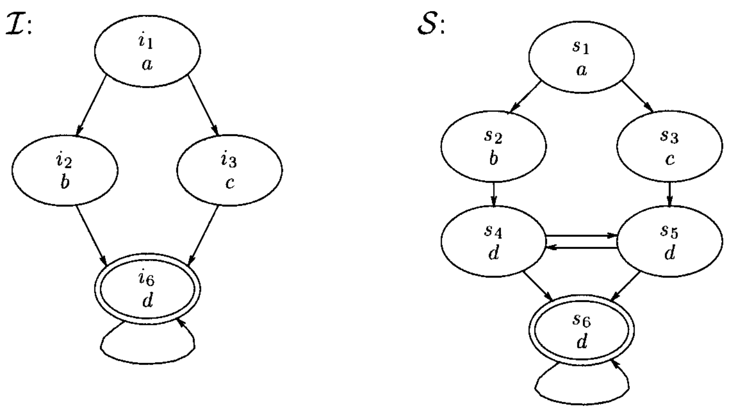


FIG. 2. ∀-simulation is stronger than fair simulation.

- Based on the locality in the game-theoretic definition of fair simulation, for two structures $\mathcal{I}$ and $\mathcal{S}$ with weak (Büchi) or strong (Streett) fairness constraints, it can be checked in time polynomial in $\mathcal{I}$ and $\mathcal{S}$ whether $\mathcal{S}$ fairly simulates $\mathcal{I}$. The algorithm, which employs tree automata, is presented in Section 4.
- Since fair simulation captures fair tree containment, it allows a logical characterization: $\mathcal{S}$ fairly simulates $\mathcal{I}$ iff every Fair-∀AFMC formula that holds in $\mathcal{S}$ holds also in $\mathcal{I}$. This is shown in Section 5.
- Fair simulation implies fair trace containment and thus provides an efficiently computable sufficient condition for checking fair trace containment. There is evidence that many practical specifications ∀-simulate their implementations. Since fair simulation is implied by ∀-simulation, the fair-simulation condition can be used as an efficient check for verifying distributed protocols that have been verified using ∀-simulation [21, 22, 24].
- In the degenerate case of vacuous fairness constraints, fair simulation coincides with simulation. In the degenerate case of deterministic systems, fair simulation coincides with fair trace containment.

We note that in process algebra, several other preorders and equivalences on state-transition systems have been extended to account for fairness, including failure preorders [4] and testing preorders [7, 12, 32]. From an algorithmic point of view, these preorders are closely related to (fair) trace containment, and the problems of checking them are complete for PSPACE.

2. DEFINITIONS

A (*Kripke*) *structure* is a 5-tuple $K = \langle \Sigma, W, \hat{w}, R, L \rangle$ with the following components:

- A finite alphabet Σ of observations. Usually, we have a finite set P of propositions and $\Sigma = 2^P$.
- A finite set W of states.
- An initial state $\hat{w} \in W$.
- A transition relation $R \subseteq W \times W$.
- A labeling function $L : W \to \Sigma$ that maps each state to an observation.

The structure K is *deterministic* if whenever $R(w, w_1)$ and $R(w, w_2)$ for $w_1 \neq w_2$, then $L(w_1) \neq L(w_2)$. For a state $w \in W$, a *w-run* of K is a finite or infinite sequence $\bar{w} = w_0 \cdot w_1 \cdot w_2 \dots$ of states $w_i \in W$ such that $w_0 = w$, and $R(w_i, w_{i+1})$ for all $i \geq 0$. We write $\inf(\bar{w})$ for the set of states that occur infinitely often in $\bar{w}$. A *run* of K is a $\hat{w}$-run, for the initial state $\hat{w}$. A *trace* of K is a finite or infinite sequence $\bar{\sigma} = \sigma_0 \cdot \sigma_1 \cdot \sigma_2 \dots$ of observations $\sigma_i \in \Sigma$ such that there is a run $\bar{w} = w_0 \cdot w_1 \cdot w_2 \dots$ of K, and $\sigma_i = L(w_i)$ for all $i \geq 0$; in this case we say that the run $\bar{w}$ *witnesses* the trace $\bar{\sigma}$.

A *fairness constraint* for K is a function that maps every infinite run of K to the binary set $\{fair, unfair\}$. We consider three kinds of fairness constraints:

- The *vacuous* constraint maps every infinite run of K to *fair*.
- A *Büchi* constraint F is specified by a set $F_B \subseteq W$ of states. Then, for an infinite run $\bar{w}$ of K we have $F(\bar{w}) = \mathit{fair}$ iff $\inf(\bar{w}) \cap F_B \neq \emptyset$.
- A *Streett* constraint F is specified by a set $F_S \subseteq 2^W \times 2^W$ of pairs of state sets. Then, for an infinite run $\bar{w}$ of K we have $F(\bar{w}) = \mathit{fair}$ iff for every pair $\langle l, r \rangle \in F_S$, if $\inf(\bar{w}) \cap l \neq \emptyset$ then $\inf(\bar{w}) \cap r \neq \emptyset$.

A *fair structure* $\mathcal{K} = \langle K, F \rangle$ consists of a structure K and a fairness constraint F for K. The fair structure $\mathcal{K}$ is a Büchi structure if F is a Büchi constraint, and $\mathcal{K}$ is a Streett structure if F is a Streett constraint. In particular, every Büchi structure is also a Streett structure. For a state $w \in W$, a *fair* w-run of $\mathcal{K}$ is either a finite w-run of K or an infinite w-run $\bar{w}$ of K such that $F(\bar{w}) = \mathit{fair}$. A fair run of $\mathcal{K}$ is a fair $\hat{w}$-run, for the initial state $\hat{w}$. A *fair trace* of $\mathcal{K}$ is a trace of K that is witnessed by a fair run of $\mathcal{K}$.

In the following, we consider two structures $K_1 = \langle \Sigma, W_1, \hat{w}_1, R_1, L_1 \rangle$ and $K_2 = \langle \Sigma, W_2, \hat{w}_2, R_2, L_2 \rangle$ over the same alphabet and two fair structures $\mathcal{K}_1 = \langle K_1, F_1 \rangle$ and $\mathcal{K}_2 = \langle K_2, F_2 \rangle$.

Trace Containment and Fair Trace Containment

The structure K_2 *trace contains* the structure K_1 if every trace of K_1 is also a trace of K_2 (or, equivalently, iff every finite trace of K_1 is also a finite trace of K_2). The problem of checking if K_2 trace contains K_1 is complete for PSPACE [31].

The fair structure $\mathcal{K}_2$ *fairly trace contains* the fair structure $\mathcal{K}_1$ if every fair trace of $\mathcal{K}_1$ is also a fair trace of $\mathcal{K}_2$. For vacuous constraints F_1 and F_2, fair trace containment coincides with trace containment. For Büchi or Streett constraints F_1 and F_2, the problem of checking if $\mathcal{K}_2$ fairly trace contains $\mathcal{K}_1$ is complete for PSPACE [29, 30].

Simulation

A binary relation $S \subseteq W_1 \times W_2$ is a *simulation* of K_1 by K_2 if the following two conditions hold [25]:

1. If $S(w_1, w_2)$, then $L_1(w_1) = L_2(w_2)$.
2. If $S(w_1, w_2)$ and $R_1(w_1, w_1')$, then there is a state $w_2' \in W_2$ such that $R_2(w_2, w_2')$ and $S(w_1', w_2')$.

The structure K_2 *simulates* the structure K_1 if there is a simulation S of K_1 by K_2 such that $S(\hat{w}_1, \hat{w}_2)$, for the initial states $\hat{w}_1$ and $\hat{w}_2$ of K_1 and K_2. The problem of checking if K_2 simulates K_1 can be solved in time $O((|W_1| + |W_2|) \cdot (|R_1| + |R_2|))$ [5, 13]. If K_2 simulates K_1, then K_2 trace contains K_1. If K_2 is deterministic, then K_2 simulates K_1 iff K_2 trace-contains K_1.

The following three alternative definitions of simulation are equivalent to the definition above.

The Game-Theoretic View. Consider a two-player game whose positions are pairs $\langle w_1, w_2 \rangle \in W_1 \times W_2$ of states. The initial position is $\langle \hat{w}_1, \hat{w}_2 \rangle$. The game is played between an antagonist and a protagonist and it proceeds in a sequence of rounds. In each round, if $\langle w_1, w_2 \rangle$ is the current position, first the antagonist updates the first component w_1 to any R_1-successor w_1', and then the protagonist updates the second component w_2 to some R_2-successor w_2' such that $L_1(w_1') = L_2(w_2')$. If no such w_2' exists, then the protagonist loses. If the game proceeds ad infinitum, for ω rounds, then the antagonist loses. It is easy to see that K_2 simulates K_1 iff the protagonist has a winning strategy.

The Tree-Containment View. A (finite or infinite) *tree* is a nonempty set $t \subseteq \mathbb{N}^*$ such that if $xn \in t$, for $x \in \mathbb{N}^*$ and $n \in \mathbb{N}$, then $x \in t$ and $xm \in t$ for all $0 \leq m < n$. The elements of t represent nodes: the empty word ϵ is the root of t, and for each node x, the nodes of the form xn, for $n \in \mathbb{N}$, are the children of x. The number of children of the node x is denoted by $deg(x)$. A *path* ρ of t is a finite or infinite set $\rho \subseteq t$ of nodes that satisfies the following three conditions: (1) $\epsilon \in \rho$, (2) for each node $x \in \rho$, there exists at most one $n \in \mathbb{N}$ with $xn \in \rho$, and (3) if $xn \in \rho$, then $x \in \rho$. Given a set A, an *A-labeled tree* is a pair $\langle t, \lambda \rangle$, where t is a tree and $\lambda : t \to A$ is a labeling function that maps each node of t to an element in A. Then, every path $\rho = \{\epsilon, n_0, n_0n_1, n_0n_1n_2, \ldots\}$ of t generates a sequence $\lambda(\rho) = \lambda(\epsilon) \cdot \lambda(n_0) \cdot \lambda(n_0n_1) \ldots$ of elements in A.

Consider a structure $K = \langle \Sigma, W, \hat{w}, R, L \rangle$. A W-labeled tree $\langle t, \lambda \rangle$ is a *run-tree* of K if $\lambda(\epsilon) = \hat{w}$, and for all nodes $x \in t$, if $xn \in t$ then $R(\lambda(x), \lambda(xn))$. A Σ-labeled tree $\langle t', \lambda' \rangle$ is a *trace-tree* of K

if there is a run-tree $\langle t, \lambda\rangle$ of K such that $t' = t$ and $\lambda' = \lambda \circ L$ (that is, for every node $x \in t$, we have $\lambda'(x) = L(\lambda(x))$); in this case we say that the run-tree $\langle t, \lambda\rangle$ *witnesses* the trace-tree $\langle t', \lambda'\rangle$. It is easy to see that K_2 simulates K_1 iff every trace-tree of K_1 is also a trace-tree of K_2 (or, equivalently, iff every finite trace-tree of K_1 is also a finite trace-tree of K_2).

The Temporal-Logic View. The three branching-time logics ∀CTL, ∀CTL*, and ∀AFMC are the fragments of CTL, CTL*, and the alternation-free μ-calculus that, in negation-free normal form (where negation can only occur in front of atomic propositions), do not contain existential path quantifiers [3, 11]. It is well known that K_2 simulates K_1 iff for every formula ψ of ∀CTL (or ∀CTL* or ∀AFMC), if K_2 satisfies ψ, then K_1 satisfies ψ.[5] It follows that similarity is the coarsest abstraction that preserves any of these three logics: For a structure K, two states w_1 and w_2 of K are *similar* if there is a simulation S such that $S(w_1, w_2)$ and a simulation S' such that $S'(w_2, w_1)$.

Previous Definitions of Fair Simulation

In the literature, we find several extensions of simulation that account for fairness constraints. In particular, the following two extensions have been studied and used extensively.

∃-Simulation [11]. A binary relation $S \subseteq W_1 \times W_2$ is an ∃-*simulation* of $\mathcal{K}_1$ by $\mathcal{K}_2$ if the following two conditions hold:

1. If $S(w_1, w_2)$, then $L_1(w_1) = L_2(w_2)$.
2. If $S(w_1, w_2)$, then for every fair w_1-run $\bar{w} = u_0 \cdot u_1 \cdot u_2 \ldots$ of $\mathcal{K}_1$, there is a fair w_2-run $\bar{w}' = u'_0 \cdot u'_1 \cdot u'_2 \ldots$ of $\mathcal{K}_2$ such that $\bar{w}'$ S-*matches* $\bar{w}$; that is, $|\bar{w}'| = |\bar{w}|$ and $S(u_i, u'_i)$ for all $0 \leq i < |\bar{w}|$.

Clearly, every ∃-simulation of $\mathcal{K}_1$ by $\mathcal{K}_2$ is a simulation of K_1 by K_2. We say that the fair structure $\mathcal{K}_2$ ∃-*simulates* the fair structure $\mathcal{K}_1$ if there is an ∃-simulation S of $\mathcal{K}_1$ by $\mathcal{K}_2$ such that $S(\hat{w}_1, \hat{w}_2)$.

For vacuous constraints, ∃-simulation coincides with simulation. For Büchi or Streett constraints F_1 and F_2, the problem of checking if $\mathcal{K}_2$ ∃-simulates $\mathcal{K}_1$ is complete for PSPACE [18]. ∃-similarity is the coarsest abstraction that preserves Fair-∀CTL*, where the universal path quantifiers range over the fair runs only: $\mathcal{K}_2$ ∃-simulates $\mathcal{K}_1$ iff for every formula ψ of Fair-∀CTL*, if $\mathcal{K}_2$ satisfies ψ, then $\mathcal{K}_1$ satisfies ψ [11]. By contrast, ∃-similarity is not the coarsest abstraction that preserves Fair-∀CTL: there are two Büchi structures $\mathcal{K}_1$ and $\mathcal{K}_2$, such that $\mathcal{K}_1$ satisfies every ∀CTL formula satisfied by $\mathcal{K}_2$, but $\mathcal{K}_2$ does not ∃-simulate $\mathcal{K}_1$ [2].[6] In addition, ∃-similarity does not preserve Fair-∀AFMC: as we show in Section 5, there are two Büchi structures $\mathcal{K}_1$ and $\mathcal{K}_2$, and a Fair-∀AFMC formula ψ, such that $\mathcal{K}_2$ ∃-simulates $\mathcal{K}_1$, and $\mathcal{K}_2$ satisfies ψ, but $\mathcal{K}_1$ does not satisfy ψ.

∀-Simulation [10, 23]. A binary relation $S \subseteq W_1 \times W_2$ is a ∀-*simulation* of $\mathcal{K}_1$ by $\mathcal{K}_2$ if the following two conditions hold:

1. S is a simulation of K_1 by K_2.
2. If $S(w_1, w_2)$, then for every fair w_1-run $\bar{w}$ of $\mathcal{K}_1$ and every w_2-run $\bar{w}'$ of K_2, if $\bar{w}'$ S-matches $\bar{w}$, then $\bar{w}'$ is a fair w_2-run of $\mathcal{K}_2$.

If S is a ∀-simulation of $\mathcal{K}_1$ by $\mathcal{K}_2$, then Condition 1 implies the following: (1) $S(w_1, w_2)$ implies $L_1(w_1) = L_2(w_2)$; (2) if $S(w_1, w_2)$, and $\bar{w}$ is a fair w_1-run of $\mathcal{K}_1$, then there exists a w_2-run $\bar{w}'$ of $\mathcal{K}_2$ that S-matches $\bar{w}$. By Condition 2, the run $\bar{w}'$ must be fair for $\mathcal{K}_2$. Thus, every ∀-simulation of $\mathcal{K}_1$ by $\mathcal{K}_2$ is an ∃-simulation of $\mathcal{K}_1$ by $\mathcal{K}_2$. We say that the fair structure $\mathcal{K}_2$ ∀-*simulates* the fair structure $\mathcal{K}_1$ if there is a ∀-simulation S of $\mathcal{K}_2$ by $\mathcal{K}_1$ such that $S(\hat{w}_1, \hat{w}_2)$.

For Büchi or Streett constraints F_1 and F_2, the problem of checking whether $\mathcal{K}_2$ ∀-simulates $\mathcal{K}_1$ is NP-complete [16]. ∀-simulation is widely used for proving abstraction hierarchies of distributed protocols [24]. In practice, Condition 2 is often replaced by a stronger condition that relates the two fairness constraints: for example, if F_1 and F_2 are both Büchi constraints, then a sufficient condition for Condition 2 is that if $S(w_1, w_2)$ and $w_1 \in F_1$, then $w_2 \in F_2$. Particularly popular is a

[5] In fact, if K_2 does not simulate K_1, then there is a formula ψ such that the only temporal modality in ψ is $\forall\bigcirc$, K_2 satisfies ψ, and K_1 does not satisfy ψ.

[6] The coarsest abstraction that preserves Fair-∀CTL is defined in [2]. The complexity of the problem of checking if $\mathcal{K}_1$ simulates $\mathcal{K}_2$ according to this definition is open.

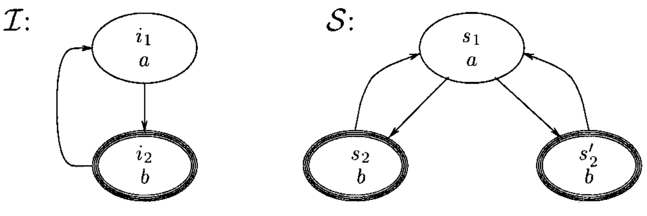


FIG. 3. Fair simulation may not have a memoryless witnessing strategy.

functional version of simulation: the simulation S is a *refinement mapping* if whenever $S(w_1, w_2)$ and $S(w_1, w'_2)$, then $w_2 = w'_2$ [1]. If S is a refinement mapping, then S is a $\forall$-simulation iff S is an $\exists$-simulation.

Our Definition of Fair Simulation

Recall the simulation game of the protagonist K_2 against the antagonist K_1. A *strategy* τ for the protagonist is a partial function from $(W_1 \times W_2)^* \times W_1$ to W_2: if the game so far has produced the sequence $\pi \in (W_1 \times W_2)^*$ of positions, and the antagonist moves to w, then the strategy τ instructs the protagonist to move to $w' = \tau(\pi, w)$, thus resulting in the new position $\langle w, w' \rangle$. Given a finite or infinite sequence $\bar{w} = u_0 \cdot u_1 \cdot u_2 \ldots$ of states $u_i \in W_1$, the *outcome* $\tau[\bar{w}] = u'_0 \cdot u'_1 \cdot u'_2 \ldots$ of the strategy τ is the finite or the infinite sequence of states $u'_i \in W_2$ such that $|\tau[\bar{w}]| = |\bar{w}|$ and $w'_i = \tau(\langle u_0, u'_0 \rangle, \langle u_1, u'_1 \rangle, \ldots, \langle u_{i-1}, u'_{i-1} \rangle, u_i)$ for all $i \geq 0$.

A binary relation $S \subseteq W_1 \times W_2$ is a *fair simulation* of $\mathcal{K}_1$ by $\mathcal{K}_2$ if the following two conditions hold:

1. If $S(w_1, w_2)$, then $L_1(w_1) = L_2(w_2)$.

2. There exists a strategy τ such that if $S(w_1, w_2)$ and $\bar{w}$ is a fair w_1-run of $\mathcal{K}_1$, the outcome $\tau[\bar{w}]$ is a fair w_2-run of $\mathcal{K}_2$ and $\tau[\bar{w}]$ S-matches $\bar{w}$. We say that τ is a *witness* to the fair simulation S.

We say that the fair structure $\mathcal{K}_2$ *fairly simulates* the fair structure $\mathcal{K}_1$ if there is a fair simulation S of $\mathcal{K}_1$ by $\mathcal{K}_2$ such that $S(\hat{w}_1, \hat{w}_2)$. For vacuous fairness constraints, fair simulation, $\exists$-simulation, and $\forall$-simulation all coincide with simulation.

In Section 4, we suggest an algorithm for checking whether one fair structure fairly simulates another. The algorithm reduces the fair-simulation problem to the nonemptiness problem of tree automata. Known results about tree automata [27, 28] then imply that in Condition 2 above, if a witnessing strategy exists, then there exists a *finite-state* witnessing strategy; that is, a strategy produced by a finite-state machine. Moreover, for Büchi structures, there exists a *memoryless* strategy, that is, a strategy that decides its next move based only on the current position of the game and the current move of the antagonist. On the other hand, for Streett structures, there may not exist a memoryless strategy. To see this, consider the two Streett structures shown below: the infinite fair runs of $\mathcal{I}$ are those that visit i_2 infinitely often, and the infinite fair runs of $\mathcal{S}$ are those that visit both s_2 and s'_2 infinitely often (i.e., $\mathcal{S}$ has the Streett constraint $\{\langle \{s_1\}, \{s_2\} \rangle, \langle \{s_1\}, \{s'_2\} \rangle\}$). In order to satisfy Condition 2, the protagonist must visit both s_2 and s'_2 infinitely often. Hence, it cannot follow a memoryless strategy.

Clearly, every fair simulation of $\mathcal{K}_1$ by $\mathcal{K}_2$ is an $\exists$-simulation of $\mathcal{K}_1$ by $\mathcal{K}_2$, and every $\forall$-simulation of $\mathcal{K}_1$ by $\mathcal{K}_2$ is a fair simulation of $\mathcal{K}_1$ by $\mathcal{K}_2$. As we demonstrated in Section 1, fair simulation falls strictly between $\exists$-simulation and $\forall$-simulation. Hence the following two propositions.

PROPOSITION 2.1. *For all fair structures $\mathcal{K}_1$ and $\mathcal{K}_2$, if $\mathcal{K}_2$ fairly simulates $\mathcal{K}_1$, then $\mathcal{K}_2$ $\exists$-simulates $\mathcal{K}_1$. There are two Büchi structures $\mathcal{K}_1$ and $\mathcal{K}_2$ such that $\mathcal{K}_2$ $\exists$-simulates $\mathcal{K}_1$, but $\mathcal{K}_2$ does not fairly simulate $\mathcal{K}_1$.*

PROPOSITION 2.2. *For all fair structures $\mathcal{K}_1$ and $\mathcal{K}_2$, if $\mathcal{K}_2$ $\forall$-simulates $\mathcal{K}_1$, then $\mathcal{K}_2$ fairly simulates $\mathcal{K}_1$. There are two Büchi structures $\mathcal{K}_1$ and $\mathcal{K}_2$ such that $\mathcal{K}_2$ fairly simulates $\mathcal{K}_1$, but $\mathcal{K}_2$ does not $\forall$-simulate $\mathcal{K}_1$.*

Fair trace containment is weaker than $\exists$-simulation [11]. However, for deterministic structures, all the definitions of fair simulation collapse and coincide with trace containment.

PROPOSITION 2.3. *For all structures $\mathcal{K}_1$ and $\mathcal{K}_2$, if $\mathcal{K}_2$ is deterministic, then the following four statements are equivalent*: (1) $\mathcal{K}_2$ *fairly trace-contains* $\mathcal{K}_1$; (2) $\mathcal{K}_2$ $\exists$*-simulates* $\mathcal{K}_1$; (3) $\mathcal{K}_2$ *fairly simulates* $\mathcal{K}_1$; (4) $\mathcal{K}_2$ $\forall$*-simulates* $\mathcal{K}_1$.

Proof. Following Propositions 2.1 and 2.2, we only need to prove that (1) implies (4). Suppose $\mathcal{K}_2$ fairly trace-contains $\mathcal{K}_1$. Assume that (1) holds. Then, for every fair run $\bar{w} = u_0 \cdot u_1 \cdot u_2 \ldots$ of $\mathcal{K}_1$, there is a fair run $\bar{w}' = u'_0 \cdot u'_1 \cdot u'_2 \ldots$ of $\mathcal{K}_2$ such that $L_1(\bar{w}) = L_2(\bar{w}')$. Define a relation $S \subseteq W_1 \times W_2$, such that $S(s, s')$ iff there exist runs $\bar{w} = u_0 \cdot u_1 \cdot u_2 \cdots s$ of $\mathcal{K}_1$ and $\bar{w}' = u'_0 \cdot u'_1 \cdot u'_2 \cdots s'$ of $\mathcal{K}_2$ such that $L_1(\bar{w}) = L_2(\bar{w}')$. Since K_2 is deterministic, for every run $\bar{w} = u_0 \cdot u_1 \cdot u_2 \ldots$ of $\mathcal{K}_1$, there is a unique $\mathcal{K}_2$ run $\bar{w}' = u'_0 \cdot u'_1 \cdot u'_2 \ldots$ of $\mathcal{K}_2$ such that for all $i \geq 0$, we have $S(u_i, u'_i)$. Further, by definition of S, we have that $\bar{w}'$ is a fair $\mathcal{K}_2$ run whenever $\bar{w}$ is a fair $\mathcal{K}_1$ run. Thus, S is a $\forall$-simulation. Further, S contains the pair of initial states $(\hat{w}_1, \hat{w}_2)$. Thus, $\mathcal{K}_2$ $\forall$-simulates $\mathcal{K}_1$. ■

As with simulation, there are three alternative definitions of fair simulation that are equivalent to the definition above.

The Game-Theoretic View. If the simulation game is played for ω rounds, then the protagonist wins. In that case, the antagonist produces an infinite run of K_1 and the protagonist produces an infinite run of K_2. In a fair game, the winning condition is modified as follows: if the game is played for ω rounds, then the protagonist wins iff either the antagonist does not produce a fair run of $\mathcal{K}_1$ or the protagonist produces a fair run of $\mathcal{K}_2$. It is easy to see that $\mathcal{K}_2$ fairly simulates $\mathcal{K}_1$ iff the protagonist has a winning strategy in the fair game.

The Tree-Containment View. Given a fair structure $\mathcal{K} = \langle K, F\rangle$, a *fair run-tree* of $\mathcal{K}$ is a run-tree $\langle t, \lambda\rangle$ of K such that every path of t generates a fair run of $\mathcal{K}$. A *fair trace-tree* of $\mathcal{K}$ is a trace-tree of K that is witnessed by a fair run-tree of $\mathcal{K}$. The following proposition gives a fully abstract tree semantics to fair simulation.

PROPOSITION 2.4. *A fair structure $\mathcal{K}_2$ fairly simulates a fair structure $\mathcal{K}_1$ iff every fair trace-tree of $\mathcal{K}_1$ is also a fair trace-tree of $\mathcal{K}_2$.*

Proof. Suppose $\mathcal{K}_2$ fairly simulates $\mathcal{K}_1$. That is, there exists a strategy τ and a relation $S \subseteq W_1 \times W_2$ such that $S(\hat{w}_1, \hat{w}_2)$, and Conditions 1 and 2 of the definition of fair simulation are satisfied. Consider a fair trace-tree $\langle t, \lambda'_1\rangle$ of $\mathcal{K}_1$ and its witnessing run-tree $\langle t, \lambda_1\rangle$. Using τ, we can construct a run-tree $\langle t, \lambda_2\rangle$ by defining λ_2 inductively as follows: (1) $\lambda_2(\epsilon) = \hat{w}_2$, and (2) for every node $n_0 n_1 n_2 \ldots n_i \in t$, the label $\lambda_2(n_0 n_1 n_2 \ldots n_{i-1} n_i)$ is given by $\tau(\langle \hat{w}_1, \hat{w}_2\rangle, \langle \lambda_1(n_0), \lambda_2(n_0)\rangle, \ldots, \langle \lambda_1(n_0 n_1 \ldots n_{i-1}), \lambda_2(n_0 n_1 \ldots n_{i-1})\rangle, \lambda_1(n_0 n_1 \ldots n_i))$. Note that $\langle t, \lambda_2\rangle$ is a fair run-tree of $\mathcal{K}_2$, because τ is a witness to a fair simulation. We can now construct the fair trace-tree $\langle t, \lambda_2 \circ L_2\rangle$ of $\mathcal{K}_2$, which is identical to $\langle t, \lambda_1\rangle$.

Conversely, suppose every fair trace-tree of $\mathcal{K}_1$ is also a fair trace-tree of $\mathcal{K}_2$. Construct a maximal fair run-tree $\langle t, \lambda_1\rangle$ of $\mathcal{K}_1$ (i.e., every fair run of $\mathcal{K}_1$ is a path in $\langle t, \lambda_1\rangle$). The corresponding fair trace-tree $\langle t, \lambda_1 \circ L_1\rangle$ is also a fair trace-tree of $\mathcal{K}_2$. Hence, it has a witnessing fair run-tree $\langle t, \lambda_2\rangle$ of $\mathcal{K}_2$. Consider the relation

$$S = \{(w_1, w_2) | \quad \text{for some } x \in t, \quad \text{we have } w_1 = \lambda_1(x) \quad \text{and} \quad w_2 = \lambda_2(x)\}.$$

We claim that S is a fair simulation of $\mathcal{K}_1$ by $\mathcal{K}_2$. We can construct the required strategy τ inductively as follows: (1) $\tau(\hat{w}_1) = \hat{w}_2$, and (2) for every run $\hat{w}_1 \cdot \lambda_1(n_0) \cdot \lambda_1(n_0 n_1) \cdots \lambda_1(n_0 n_1 \ldots n_i)$, we have

$$\tau(\langle \hat{w}_1, \hat{w}_2\rangle, \langle \lambda_1(n_0), \lambda_2(n_0)\rangle, \ldots, \lambda_1(n_0 n_1 \ldots n_{i-1} n_i)) = \lambda_2(n_0 n_1 n_2 \ldots n_i).$$

Since $\langle t, \lambda_2\rangle$ is a fair run-tree of $\mathcal{K}_2$, the strategy τ is a witnessing strategy for S. ■

The Temporal-Logic View. In Section 5, we will show that $\mathcal{K}_2$ fairly simulates $\mathcal{K}_1$ iff for every formula ψ of Fair-$\forall$AFMC, if $\mathcal{K}_2$ satisfies ψ, then $\mathcal{K}_1$ satisfies ψ. It follows that fair similarity is the coarsest abstraction that preserves the fair universal alternation-free μ-calculus: For a fair structure $\mathcal{K}$, two states w_1 and w_2 of $\mathcal{K}$ are *fairly similar* if there is a fair simulation S such that $S(w_1, w_2)$ and a fair simulation S' such that $S'(w_2, w_1)$.

3. INITIALIZED SIMULATIONS

In this section, we consider weak versions of ∃-simulation, ∀-simulation, and fair simulation, where the S-matching is restricted to fair runs that start at the initial states of $\mathcal{K}_1$ and $\mathcal{K}_2$. We refer to the weak versions as init-∃-simulation, init-∀-simulation, and init-fair simulation, and we investigate two properties of such init-similarities:

- *Monotonicity*: given a relation S that is an init-∃-simulation (init-∀-simulation, init-fair simulation) and a finite simulation $S' \supseteq S$, is S' guaranteed to be an init-∃-simulation (init-∀-simulation, init-fair simulation) as well?
- *Initial-sensitivity*: does the existence of an init-∃-simulation (init-∀-simulation, init-fair simulation) from $\mathcal{K}_1$ to $\mathcal{K}_2$ guarantee the existence of an ∃-simulation (∀-simulation, fair simulation)?

As we shall see in this section, fair simulation enjoys both monotonicity and initial-sensitivity. This leads us, as detailed in Section 4, to an efficient algorithm for checking fair simulation between two fair structures. In addition, the lack of either monotonicity or initial-sensitivity for init-∃-simulation and init-∀-simulation gives some intuition for why there is they are computationally hard to check (no polynomial algorithm is known).

We first define init-∃-simulation, init-∀-simulation, and init-fair simulation formally. As in the previous section, we consider two structures over the same alphabet, $K_1 = \langle \Sigma, W_1, \hat{w}_1, R_1, L_1 \rangle$ and $K_2 = \langle \Sigma, W_2, \hat{w}_2, R_2, L_2 \rangle$, and two fair structures, $\mathcal{K}_1 = \langle K_1, F_1 \rangle$ and $\mathcal{K}_2 = \langle K_2, F_2 \rangle$.

Init-∃-Simulation. A binary relation $S \subseteq W_1 \times W_2$ is an *init-∃-simulation* of $\mathcal{K}_2$ by $\mathcal{K}_1$ if the following three conditions hold:

1. If $S(w_1, w_2)$, then $L_1(w_1) = L_2(w_2)$.
2. For every fair $\hat{w}_1$-run $\bar{w}$ of $\mathcal{K}_1$, there exists an S-matching fair $\hat{w}_2$-run $\bar{w}'$ of $\mathcal{K}_2$.

Init-∀-Simulation. A binary relation $S \subseteq W_1 \times W_2$ is an *init-∀-simulation* of $\mathcal{K}_2$ by $\mathcal{K}_1$ if the following two conditions hold:

1. S is a simulation of K_2 by K_1.
2. For every fair $\hat{w}_1$-run $\bar{w}$ of $\mathcal{K}_1$ and every $\hat{w}_2$-run $\bar{w}'$ of $\mathcal{K}_2$, if $\bar{w}'$ S-matches $\bar{w}$, then $\bar{w}'$ is a fair run of $\mathcal{K}_2$.

Init-Fair-Simulation. A binary relation $S \subseteq W_1 \times W_2$ is an *init-fair simulation* of $\mathcal{K}_2$ by $\mathcal{K}_1$ if the following two conditions hold:

1. If $S(w_1, w_2)$, then $L_1(w_1) = L_2(w_2)$.
2. There exists a strategy τ such that for every fair $\hat{w}_1$-run $\bar{w}$ of $\mathcal{K}_1$, the outcome $\tau[\bar{w}]$ is a fair $\hat{w}_2$-run of $\mathcal{K}_2$ and $\tau[\bar{w}]$ S-matches $\bar{w}$.

As with noninitialized simulation, init-fair-simulation falls strictly between init-∃-simulation and init-∀-simulation.

We first consider the monotonicity of the initialized simulations.

PROPOSITION 3.1. *For all fair structures $\mathcal{K}_1 = \langle K_1, F_1 \rangle$ and $\mathcal{K}_2 = \langle K_2, F_2 \rangle$, if S is an init-∃-simulation (init-fair simulation) of $\mathcal{K}_1$ by $\mathcal{K}_2$, and $S' \supseteq S$ is a simulation of K_1 by K_2, then S' is also an init-∃-simulation (init-fair simulation) of $\mathcal{K}_1$ by $\mathcal{K}_2$.*

Proof. Let S be an init-∃-simulation from $\mathcal{K}_1$ to $\mathcal{K}_2$. Given a fair run $\bar{w}$ of $\mathcal{K}_1$, we know that there exists an S-matching fair run $\bar{w}'$ of $\mathcal{K}_2$. Since $\bar{w}'$ also S'-matches $\bar{w}$, we have that S' is an init-∃-simulation as well, and we are done.

Similarly, if S is an init-fair simulation, there is a witnessing strategy τ such that given a fair run $\bar{w}$ of $\mathcal{K}_1$, the outcome $\tau[\bar{w}]$ is a fair run of $\mathcal{K}_2$ and $\tau[\bar{w}]$ S-matches $\bar{w}$. Since $\tau[\bar{w}]$ S'-matches $\bar{w}$, we have that S' is an init-fair simulation as well, with the same strategy τ as witness.

PROPOSITION 3.2. *There are two Büchi structures $\mathcal{K}_1$ and $\mathcal{K}_2$, an init-∀-simulation S of $\mathcal{K}_1$ by $\mathcal{K}_2$, and a simulation $S' \supseteq S$ of K_1 by K_2, such that S' is not an init-∀-simulation of $\mathcal{K}_1$ by $\mathcal{K}_2$.*

Proof. Consider the structures in Fig. 3, and replace the Streett constraint of $\mathcal{S}$ with the Büchi constraint $\{s_2\}$. Thus, an infinite run of $\mathcal{S}$ is fair iff it visits the state s_2 infinitely often. Consider the relations $S = \{(i_1, s_1), (i_2, s_2)\}$ and $S' = \{(i_1, s_1), (i_2, s_2), (i_2, s_2')\}$. Both S and S' are simulations of $\mathcal{I}$ by $\mathcal{S}$, and $S' \supseteq S$. There is a single infinite fair run for $\mathcal{I}$, namely, $(i_1 \cdot i_2)^\omega$. This run has two S-matching runs of $\mathcal{S}$, namely $(s_1 \cdot s_2)^\omega$ and $(s_1 \cdot s_2')^\omega$. Of these, only $(s_1 \cdot s_2)^\omega$ is fair. Thus, while S is an init-∀-simulation of $\mathcal{I}$ by $\mathcal{S}$, the relation S' is not. ■

Next, we consider the initial-sensitivity of the initialized simulations.

Proposition 3.3. *For all fair structures $\mathcal{K}_1 = \langle K_1, F_1\rangle$ and $\mathcal{K}_2 = \langle K_2, F_2\rangle$, $\mathcal{K}_2$ init-fairly simulates (init-∀-simulates) $\mathcal{K}_1$ iff $\mathcal{K}_2$ fairly simulates (∀-simulates) $\mathcal{K}_1$.*

Proof. Let S be an init-fair simulation from $\mathcal{K}_1$ to $\mathcal{K}_2$, with strategy τ as the witness. We define

$$\begin{aligned} S' = \{\,(w, w') \mid (w, w') \in S \text{ and there exists a finite run}\\ \bar{w} = \hat{w}_1 \cdots w \text{ of } \mathcal{K}_1, \text{ with } \tau[\bar{w}] = \hat{w}_2 \cdots w'\}. \end{aligned}$$

Intuitively, S' is the subset of pairs from S that are used by the protagonist in the fair simulation game. It is easy to see that S' is a fair simulation from $\mathcal{K}_1$ to $\mathcal{K}_2$. Similarly, let S be an init-∀-simulation from $\mathcal{K}_1$ to $\mathcal{K}_2$. We define

$$\begin{aligned} S' = \{(w, w') \mid (w, w') \in S \text{ and there exist finite runs}\\ \bar{w} = \hat{w}_1 \cdots w' \text{ of } \mathcal{K}_1 \text{ and } \bar{w}' = \hat{w}_2 \cdots w \text{ of } \mathcal{K}_2,\\ \text{such that } \bar{w}'\ S - \text{matches } \bar{w}\}. \end{aligned}$$

It is easy to see that S' is a ∀-simulation from $\mathcal{K}_1$ to $\mathcal{K}_2$. ■

Before investigating the initial sensitivity of ∃-simulation, we first note the following strong relation between init-∃-simulation and fair trace containment.

Proposition 3.4. *For all fair structures $\mathcal{K}_1 = \langle K_1, F_1\rangle$ and $\mathcal{K}_2 = \langle K_2, F_2\rangle$, $\mathcal{K}_2$ fairly trace-contains $\mathcal{K}_1$ iff $\mathcal{K}_2$ init-∃-simulates $\mathcal{K}_1$.*

Proof. It is easy to see that init-∃-simulation implies fair trace containment. To prove the other direction, assume $\mathcal{K}_2$ fairly trace-contains $\mathcal{K}_1$. Then, we claim that the relation

$$S = \{(w_1, w_2) \mid w_1 \in W_1, w_2 \in W_2, \text{and } L_1(w_1) = L_2(w_2)\}$$

is an init-∃-simulation. Indeed, for every fair run of $\mathcal{K}_1$, we can produce an S-matching fair run of $\mathcal{K}_2$. ■

Proposition 3.4 and the well-known fact that, on Büchi structures, trace containment is strictly weaker than ∃-simulation [11] lead to the following proposition.

Proposition 3.5. *There are two Büchi structures $\mathcal{K}_1$ and $\mathcal{K}_2$, such that $\mathcal{K}_2$ init-∃-simulates $\mathcal{K}_1$, but $\mathcal{K}_2$ does not ∃-simulate $\mathcal{K}_1$.*

It follows (see Table 1) that only fair simulation enjoys both monotonicity and initial sensitivity. Checking if a given relation is an ∃-simulation is PSPACE-complete [18]. Checking if a given relation is a ∀-simulation can be done in PTIME, but because of a lack of monotonicity, finding a good candidate

TABLE 1

Properties of Initialized Simulations

Init simulation	Monotonicity	Init-sensitivity
∃-simulation	+	−
∀-simulation	−	+
Fair simulation	+	+

relation is hard (NP-complete [16]). In the next section, we exploit the monotonicity and init sensitivity of fair simulation to obtain an efficient algorithm for checking if there is a fair simulation from one fair structure to another.

4. CHECKING FAIR SIMULATION

Given two structures $K_1 = \langle \Sigma, W_1, \hat{w}_1, R_1, L_1 \rangle$ and $K_2 = \langle \Sigma, W_2, \hat{w}_2, R_2, L_2 \rangle$ with $W_1 \cap W_2 = \emptyset$, and two fair structures $\mathcal{K}_1 = \langle K_1, F_1 \rangle$ and $\mathcal{K}_2 = \langle K_2, F_2 \rangle$, we present an automata-based algorithm that checks, in time polynomial in K_1 and K_2, whether $\mathcal{K}_2$ fairly simulates $\mathcal{K}_1$.

A *maximal simulation* is a binary relation $\hat{S} \subseteq W_1 \times W_2$ such that (1) $\hat{S}$ is a simulation of K_1 by K_2 and (2) for every simulation S of K_1 by K_2, we have $S \subseteq \hat{S}$. The following proposition reduces the problem of checking if there is a fair simulation of $\mathcal{K}_1$ by $\mathcal{K}_2$ to checking if the (unique) maximal simulation of K_1 by K_2 is an init-fair simulation of $\mathcal{K}_1$ by $\mathcal{K}_2$.

PROPOSITION 4.1. *For all fair structures $\mathcal{K}_1 = \langle K_1, F_1 \rangle$ and $\mathcal{K}_2 = \langle K_2, F_2 \rangle$, if $\hat{S}$ is the maximal simulation of K_1 by K_2, then $\mathcal{K}_2$ fairly simulates $\mathcal{K}_1$ iff $\hat{S}$ is an init-fair simulation of $\mathcal{K}_1$ by $\mathcal{K}_2$.*

Proof. Due to initial-sensitivity (Proposition 3.3), we know $\mathcal{K}_2$ fairly simulates $\mathcal{K}_1$ iff $\mathcal{K}_2$ init-fairly simulates $\mathcal{K}_1$.

Suppose that S is an init-fair simulation of $\mathcal{K}_1$ by $\mathcal{K}_2$. Since S has to be a simulation of K_1 by K_2 as well, and $\hat{S}$ is the maximal simulation, we have $S \subseteq \hat{S}$. Further, due to monotonicity (Proposition 3.1), $\hat{S}$ has to be an init-fair simulation as well. ■

The maximal simulation of K_1 by K_2 can be constructed in time $O((|W_1| + |W_2|) \cdot (|R_1| + |R_2|))$ [5, 13]. Hence, it is left to find an algorithm that efficiently checks, given a relation $S \subseteq W_1 \times W_2$, if S is an init-fair simulation of $\mathcal{K}_1$ by $\mathcal{K}_2$. For this purpose, consider the structure $K_S = \langle \Sigma_S, W, \hat{w}, R, L \rangle$, with the following components:

- $\Sigma_S = W_1 \cup W_2$. Thus, each state of K_S is labeled by a state of K_1 or K_2.
- $W = (S \times \{\mathbf{a}\}) \cup (W_1 \times W_2 \times \{\mathbf{p}\})$. Thus, there are two types of states: antagonist states, in which the W_1-component is related by S to the W_2-component, and protagonist states, which are not restricted. We regard the states of K_S as positions in a game, with the antagonist moving in antagonist states and the protagonist moving in protagonist states.
- $\hat{w} = \langle \hat{w}_1, \hat{w}_2, \mathbf{a} \rangle$. This is the initial game position.
- $R = \{(\langle w_1, w_2, \mathbf{a} \rangle, \langle w_1', w_2, \mathbf{p} \rangle) \mid R_1(w_1, w_1')\} \cup \{(\langle w_1, w_2, \mathbf{p} \rangle, \langle w_1, w_2', \mathbf{a} \rangle) \mid R_2(w_2, w_2')\}$. Thus, the antagonist and the protagonist alternate moves. The antagonist moves along transitions that correspond to transitions of K_1, and the protagonist moves along transitions that correspond to transitions of K_2. Since antagonist states consist only of pairs in S, the protagonist must reply to each move of the antagonist to a state $\langle w_1', w_2, \mathbf{p} \rangle$ with a move to a state $\langle w_1', w_2', \mathbf{a} \rangle$ for which $S(w_1', w_2')$.
- We label each antagonist state by its W_1-component, and we label each protagonist state by its W_2-component; that is, $L(\langle w_1, w_2, \mathbf{a} \rangle) = \{w_1\}$ and $L(\langle w_1, w_2, \mathbf{p} \rangle) = \{w_2\}$.

A run $\bar{w}$ of K_S satisfies a fairness constraint F for $\mathcal{K}_1$ or $\mathcal{K}_2$ if $F(L(\bar{w})) = \mathit{fair}$. The protagonist wins the game on K_S if (1) whenever the game position is a protagonist-state, the protagonist can proceed with a move, and (2) whenever the game produces an infinite run of K_S, either the run does not satisfy F_1 or it satisfies both F_1 and F_2. Then, the protagonist has a winning strategy in this game iff S is an init-fair simulation of $\mathcal{K}_1$ by $\mathcal{K}_2$.

The problem of checking the existence of a winning strategy (and the synthesis of such a strategy [27, 28]) can be reduced to the nonemptiness problem for tree automata. We first define finite automata over words and trees formally.

Word Automata. An automaton on (finite or infinite) words is a quintuple $\mathcal{A} = \langle \Sigma, Q, \hat{q}, \delta, \alpha \rangle$, with the following components:

- A finite alphabet Σ.
- A finite set Q of states.

- An initial state $\hat{q} \in Q$.
- A transition function $\delta : Q \times \Sigma \to 2^Q$.
- An acceptance condition $\alpha : Q^* \cup Q^\omega \to \{accepting, rejecting\}$. The condition α classifies the runs of $\mathcal{A}$ (we define runs shortly) to accepting and rejecting.

A run of $\mathcal{A}$ on an input word $\bar{\sigma} = \sigma_0 \cdot \sigma_1 \cdot \sigma_2 \ldots$ is a word $\bar{q} = q_0 \cdot q_1 \cdot q_2 \ldots$ such that $q_0 = \hat{q}$ and $q_{i+1} \in \delta(q_i, \sigma_i)$ for all $i \geq 0$. The run $\bar{q}$ is accepting if $\alpha(\bar{q}) = accepting$. The automaton accepts $\bar{\sigma}$ if there exists an accepting run of $\mathcal{A}$ on $\bar{\sigma}$. The set of all words accepted by $\mathcal{A}$ is denoted by $\mathcal{L}(\mathcal{A})$, the language of $\mathcal{A}$.

Tree Automata. An automaton on (finite or infinite) trees is a hextuple $\mathcal{A} = \langle \Sigma, Q, \hat{q}, \mathcal{D}, \delta, \alpha \rangle$, where Σ, Q, $\hat{q}$, and α are as in word automata, and in addition, we have:

- A finite set $\mathcal{D} \subset \mathbb{N}$ of possible branching degrees.
- A transition function $\delta : Q \times \Sigma \times \mathcal{D} \to 2^{Q^*}$ such that for every $q \in Q$, $\sigma \in \Sigma$, and $k \in \mathcal{D}$, we have $\delta(q, \sigma, k) \subseteq Q^k$. Thus, both σ and k are arguments of the transition function.

A run of $\mathcal{A}$ on an input Σ-labeled tree $\langle t, \lambda \rangle$ is a Q-labeled tree $\langle t, \theta \rangle$ such that $\theta(\epsilon) = \hat{q}$ and for every $x \in t$, we have

$$\langle \theta(x \cdot 0), \theta(x \cdot 1), \ldots, \theta(x \cdot (deg(x) - 1)) \rangle \in \delta(\theta(x), \lambda(x), deg(x)).$$

If, for instance, $\theta(0) = q_0$, $\lambda(0) = \sigma$, $deg(0) = 2$, and $\delta(q_0, \sigma, 2) = \{\langle q_1, q_2 \rangle, \langle q_4, q_5 \rangle\}$, then either $\theta(0 \cdot 0) = q_1$ and $\theta(0 \cdot 1) = q_2$ or $\theta(0 \cdot 0) = q_4$ and $\theta(0 \cdot 1) = q_5$. A run $\langle t, \theta \rangle$ is accepting if $\alpha(\theta(\rho)) = accepting$ for all maximal paths $\rho \subseteq t$ (a path ρ is maximal if ρ is infinite or if ρ is finite yet there is no node x such that $\rho \cup \{x\}$ is a path in t). The automaton $\mathcal{A}$ accepts $\langle t, \lambda \rangle$ if there exists an accepting run of $\mathcal{A}$ on $\langle t, \lambda \rangle$. The set of all trees accepted by $\mathcal{A}$ is denoted by $\mathcal{L}(\mathcal{A})$, the language of $\mathcal{A}$.

We now return to the problem of deciding whether the protagonist has a winning strategy in the game played on K_S. We construct two tree automata:

1. The tree automaton $\mathcal{A}_S$ accepts all $(W_1 \cup W_2)$-labeled trees that can be obtained by unrolling K_S and pruning from it subtrees that have as a root an antagonist state, so that each protagonist state has exactly one successor. The intuition is that each tree accepted by $\mathcal{A}_S$ corresponds to a strategy of the protagonist.

The automaton $\mathcal{A}_S$ has $|W_1| \cdot |W_2|$ states, and it has a vacuous acceptance condition. Formally, $\mathcal{A}_S = \langle \Sigma_S, W, \hat{w}, \mathcal{D}, \delta, \alpha_S \rangle$, where $\mathcal{A}_S$ has the same input alphabet as the observation alphabet of K_S and the same states and initial state as K_S. The set of possible branching degrees is given by

$$\mathcal{D} = \{k \mid k \text{ is the outdegree of an antagonist state}\} \cup \{1\}.$$

The transition function δ is defined by:

- $\delta(\langle w_1, w_2, \mathbf{a} \rangle, w_1, k) = \{\langle p_1, p_2, p_3, \ldots, p_k \rangle\}$, where k is the out degree of $\langle w_1, w_2, \mathbf{a} \rangle$ and $R(\langle w_1, w_2, \mathbf{a} \rangle, p_i)$ for $i = 1, 2, \ldots, k$.
- $\delta(\langle w_1, w_2, \mathbf{p} \rangle, w_2, 1) = \{\langle a_1 \rangle, \langle a_2 \rangle, \langle a_3 \rangle, \ldots, \langle a_k \rangle\}$, where k is the out-degree of $\langle w_1, w_2, \mathbf{p} \rangle$ and $R(\langle w_1, w_2, \mathbf{p} \rangle, a_i)$ for $i = 1, 2, \ldots, k$.

The acceptance condition α_S maps all sequences in $W^* \cup W^\omega$ to *accepting*.

2. The tree automaton $\mathcal{A}_F$ accepts all trees labeled by $W_1 \cup W_2$ in which all paths that satisfy F_1 satisfy F_2 as well. In order to define $\mathcal{A}_F$, we first construct a deterministic word automaton $\mathcal{A}_W$. The automaton $\mathcal{A}_W$ accepts words over $W_1 \cup W_2$ satisfying the LTL formula $\psi = (\phi_1 \to \phi_2)$, where ϕ_1 and ϕ_2 are LTL formulas corresponding to F_1 and F_2, respectively (we consider here Büchi and Streett automata and thus F_1 and F_2 indeed have corresponding LTL formulas). We use usual notation to describe LTL formulas. For example, an infinite run $\bar{w}$ satisfies $\Box \Diamond p$ if p is true in infinitely many states of $\bar{w}$, and it satisfies $\Diamond \Box p$ if p is true in all states of some infinite suffix of $\bar{w}$. The definition of $\mathcal{A}_W$ depends on the type of the fairness constraints of $\mathcal{K}_1$ and $\mathcal{K}_2$:

- When $\mathcal{K}_1$ and $\mathcal{K}_2$ are Büchi structures, ψ has the form $\Box\Diamond l \to \Box\Diamond r$, where $l = F_1 \subseteq W_1$ and $r = F_2 \subseteq W_2$. Then,

$$\mathcal{A}_W = \langle \Sigma_S, \{q_0, q_l, q_r\}, q_0, \delta, \{\langle\{q_l\}, \{q_r\}\rangle\}\rangle,$$

where for every $\varsigma \in \Sigma_S$, we have

$$\delta(q_0, \varsigma) = \delta(q_l, \varsigma) = \delta(q_r, \varsigma) = \begin{cases} q_l & \text{if } \varsigma \in l, \\ q_r & \text{if } \varsigma \in r, \\ q_0 & \text{otherwise.} \end{cases}$$

Note that the acceptance condition for $\mathcal{A}_W$ is a Streett condition $\{\langle\{q_l\}, \{q_r\}\rangle\}$, which states that a run is accepting if either it visits q_l only finitely many times, or it visits q_r infinitely many times.

- When $\mathcal{K}_1$ and $\mathcal{K}_2$ are Streett structures with

$$\begin{aligned} F_1 &= \{\langle l_1^1, r_1^1\rangle, \langle l_2^1, r_2^1\rangle, \ldots, \langle l_{|F_1|}^i, r_{|F_1|}^i\rangle\}, \quad \text{and} \\ F_2 &= \{\langle l_1^2, r_1^2\rangle, \langle l_2^2, r_2^2\rangle, \ldots, \langle l_{|F_2|}^2, r_{|F_2|}^2\rangle\}, \end{aligned}$$

the formula ψ has the form

$$\bigwedge_{i=1}^{|F_1|} \left(\Box\Diamond l_i^1 \to \Box\Diamond r_i^1\right) \to \bigwedge_{j=1}^{|F_2|} \left(\Box\Diamond l_j^2 \to \Box\Diamond r_j^2\right).$$

We can rewrite ψ to have $|F_1| + 1$ disjuncts, as

$$\left(\bigvee_{i=1}^{|F_1|} \neg\left(\Box\Diamond l_i^1 \to \Box\Diamond r_i^1\right)\right) \vee \bigwedge_{j=1}^{|F_2|} \left(\Box\Diamond l_j^2 \to \Box\Diamond r_j^2\right).$$

With each of the first $|F_1|$ disjuncts $\neg(\Box\Diamond l_i^1 \to \Box\Diamond r_i^1)$, we associate a word automaton

$$\mathcal{A}_i = \langle \Sigma_S, \{q_0, q_l, q_r\}, q_0, \delta, \{\langle\{q_0, q_l, q_r\}, \{q_l\}\rangle, \langle\{q_r\}, \{\}\rangle\}\rangle,$$

where for every $\varsigma \in \Sigma_S$, we have

$$\delta(q_0, \varsigma) = \delta(q_l, \varsigma) = \delta(q_r, \varsigma) = \begin{cases} q_l & \text{if } \varsigma \in l_i^1, \\ q_r & \text{if } \varsigma \in r_i^1, \\ q_0 & \text{otherwise.} \end{cases}$$

Note that the acceptance condition for $\mathcal{A}_i$ is a Streett condition with two Streett pairs. Consider any accepting run of $\mathcal{A}_i$. The first Streett pair $\langle\{q_0, q_l, q_r\}, \{q_l\}\rangle$ ensures that the run visits q_l infinitely many times. The second Streett pair $\langle\{q_r\}, \{\}\rangle$ ensures that the run visits q_r only finitely many times.

For the last disjunct of ψ we construct a word automaton $\mathcal{A}_{|F_1|+1}$ with $|F_2|$ states and $|F_2|$ Streett pairs. We then take the union of the $|F_1| + 1$ automata and obtain a deterministic Streett automaton $\mathcal{A}_W$ with $3^{|F_1|} \times |F_2|$ states and $(2 \cdot |F_1|) + |F_2|$ Streett pairs.

Since the word automaton $\mathcal{A}_W$ is deterministic, it can be transformed into tree automaton $\mathcal{A}_F$, whose branching degrees $\mathcal{D}$ are taken from $\mathcal{A}_S$. Formally, if $\mathcal{A}_W = \langle \Sigma, Q, \hat{q}, \delta, \alpha\rangle$, then $\mathcal{A}_F = \langle \Sigma, Q, \hat{q}, \mathcal{D}, \delta', \alpha\rangle$, where $\delta'(q, \varsigma, k) = \langle q', q' \ldots, q'\rangle$ whenever $k \in \mathcal{D}$, and $\delta(q, \varsigma) = q'$. Since $\mathcal{A}_W$ is deterministic, a tree $\langle t, \lambda\rangle$ is in $\mathcal{L}(\mathcal{A}_F)$ iff all branching degrees of t are in $\mathcal{D}$, and for every infinite path ρ in $\langle t, \lambda\rangle$, we have $\lambda(\rho) \in \mathcal{L}(\mathcal{A}_W)$ [17].

Note that $\mathcal{A}_F$ has the same number of states and Streett pairs as $\mathcal{A}_W$.

It is easy to see that the protagonist has a winning strategy iff the intersection of the Streett automata $\mathcal{A}_S$ and $\mathcal{A}_F$ is nonempty. To check the latter, we define and check the nonemptiness of the product automaton $\mathcal{A}_S \times \mathcal{A}_F$. Since $\mathcal{A}_S$ has a vacuous acceptance condition, the product automaton is a Streett automaton with the same number of pairs as $\mathcal{A}_F$. Finally, since checking the nonemptiness of a Streett tree automaton with n states and f pairs requires time $O(n^{(2f+1)} \cdot f!)$ [19], the theorem below follows.

THEOREM 4.1. *Given two fair structures $\mathcal{K}_1$ and $\mathcal{K}_2$ with state sets W_1 and W_2, transition relations R_1 and R_2, and fairness constraints F_1 and F_2, we can check whether $\mathcal{K}_2$ fairly simulates $\mathcal{K}_1$ in time*:

- $O((|W_1| + |W_2|) \cdot (|R_1| + |R_2|) + (|W_1| \cdot |W_2|)^3)$, *for Büchi structures.*
- $O(n^{(2f+1)} \cdot f!)$, *where $n = |W_1| \cdot |W_2| \cdot (3^{|F_1|} + |F_2|)$ and $f = 2 \cdot |F_1| + |F_2|$, for Streett structures.*

5. A LOGICAL CHARACTERIZATION OF FAIR SIMULATION

We show that fair simulation characterizes the distinguishing power of the fair universal fragment of the alternation-free μ-calculus (Fair-∀AFMC); that is, for every two fair structures $\mathcal{K}_1$ and $\mathcal{K}_2$, every Fair-∀AFMC formula that is satisfied in $\mathcal{K}_2$ is satisfied also in $\mathcal{K}_1$ iff $\mathcal{K}_2$ fairly simulates $\mathcal{K}_1$. For technical convenience, we consider ∃AFMC, the dual, existential fragment of the alternation-free μ-calculus.

Syntax and Semantics of Fair-∃AFMC

The syntax of ∃AFMC is defined with respect to a set P of propositions and a set V of variables. We first define the syntax of the entire existential μ-calculus (∃ MC). A formula of ∃ MC is one of the following:

- **true**, **false**, p, or $\neg p$, for $p \in P$.
- y, for $y \in V$.
- $\varphi_1 \vee \varphi_2$ or $\varphi_1 \wedge \varphi_2$, where φ_1 and φ_2 are ∃ MC formulas.
- $\exists\bigcirc\varphi$, where φ is an ∃ MC formula.
- $\mu y.f(y)$ or $\nu y.f(y)$, where $f(y)$ is an ∃ MC formula. All occurrences of the variable y in $\mu y.f(y)$ and $\nu y.f(y)$ are bound.

An ∃ MC formula is *alternation-free* if for all $y \in V$, there are respectively no occurrences of ν (μ) in any syntactic path from an occurrence of μy (νy) to an occurrence of y. For example, the formula $\mu x.(p \vee \mu y.(x \vee \exists\bigcirc y))$ is alternation-free, and the formula $\mu x.(p \vee \nu y.(x \vee \exists\bigcirc y))$ is not alternation-free. The existential alternation-free μ-calculus (∃AFMC) is the set of all ∃ MC formulas that are alternation-free.

The semantics of ∃AFMC is defined for formulas without free occurrences of variables. From now on we consider only such closed formulas. We interpret ∃AFMC formulas over *fair* structures, thus obtaining the logic Fair-∃AFMC. While there is no obvious interpretation of the full alternation-free μ-calculus over fair structures, ∃AFMC does not admit universal path quantification, which enables us to limit all fixed-point calculations to fair paths. Before we give the formal semantics of Fair-∃AFMC, let us note that since AFMC is a "local" logic, in the sense that its only temporal operator is next, and fairness is a global property, any attempt to add fairness to AFMC must result in something that may seem less than satisfactory. We believe that our definition is the most natural semantics one can obtain by adding fairness to a fragment of AFMC. In particular, as we shall note below, when we restrict attention to Fair-∃AFMC formulas that have equivalences of CTL, our semantics of Fair-∃AFMC coincides with the standard semantics for Fair-CTL [6].

The *closure* $cl(\psi)$ of a Fair-∃AFMC formula ψ is the least set of formulas that satisfies the following conditions:

- **true** $\in cl(\psi)$ and **false** $\in cl(\psi)$.
- $\psi \in cl(\psi)$.
- If $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$ is in $cl(\psi)$, then $\varphi_1 \in cl(\psi)$ and $\varphi_2 \in cl(\psi)$.

- If $\exists \bigcirc \varphi \in cl(\psi)$, then $\varphi \in cl(\psi)$.
- If $\mu y.f(y) \in cl(\psi)$, then $f(\mu y.f(y)) \in cl(\psi)$.
- If $\nu y.f(y) \in cl(\psi)$, then $f(\nu y.f(y)) \in cl(\psi)$.

Each Fair-∃AFMC formula ψ specifies a set of "obligations"—a subset of formulas in $cl(\psi)$—that need to be satisfied. The witness to the satisfaction of a formula is a tree called a sat-tree. Formally, given a fair structure $\mathcal{K} = \langle K, F\rangle$ with $K = \langle \Sigma, W, w, R, L\rangle$, and a Fair-∃AFMC formula ψ, a *sat-tree* $\langle t, \lambda\rangle$ of $\mathcal{K}$ for ψ is a $(W \times cl(\psi))$-labeled tree $\langle t, \lambda\rangle$ that satisfies the following conditions:

- $\lambda(\epsilon) = \langle \hat{w}, \psi\rangle$. Thus, the root of the tree, which corresponds to the initial obligation, is labeled by the initial state of K and ψ itself.
- If $\lambda(x) = \langle w, \mathbf{false}\rangle$, then $deg(x) = 0$.
- If $\lambda(x) = \langle w, \mathbf{true}\rangle$ and w has no successors in K, then $deg(x) = 0$.
- If $\lambda(x) = \langle w, \mathbf{true}\rangle$ and w has successors in K, then $deg(x) = 1$ and $\lambda(x0) \in \{\langle w', \mathbf{true}\rangle \mid R(w, w')\}$.
- If $\lambda(x) = \langle w, p\rangle$, where $p \in P$, then $deg(x) = 1$. If $p \in L(w)$, then $\lambda(x0) = \langle w, \mathbf{true}\rangle$; otherwise $\lambda(x0) = \langle w, \mathbf{false}\rangle$.
- If $\lambda(x) = \langle w, \neg p\rangle$, where $p \in P$, then $deg(x) = 1$. If $p \in L(w)$, then $\lambda(x0) = \langle w, \mathbf{false}\rangle$; otherwise $\lambda(x0) = \langle w, \mathbf{true}\rangle$.
- If $\lambda(x) = \langle w, \varphi_1 \vee \varphi_2\rangle$, then $deg(x) = 1$ and $\lambda(x0) \in \{\langle w, \varphi_1\rangle, \langle w, \varphi_2\rangle\}$.
- If $\lambda(x) = \langle w, \varphi_1 \wedge \varphi_2\rangle$, then $deg(x) = 2$, $\lambda(x0) = \langle w, \varphi_1\rangle$, and $\lambda(x1) = \langle w, \varphi_2\rangle$.
- If $\lambda(x) = \langle w, \exists \bigcirc \varphi\rangle$, then $deg(x) = 1$ and $\lambda(x0) \in \{\langle w', \varphi\rangle \mid R(w, w')\}$.
- If $\lambda(x) = \langle w, \nu y.f(y)\rangle$, then $deg(x) = 1$ and $\lambda(x0) = \langle w, f(\nu y.f(y))\rangle$.
- If $\lambda(x) = \langle w, \mu y.f(y)\rangle$, then $deg(x) = 1$ and $\lambda(x0) = \langle w, f(\mu y.f(y))\rangle$.

Consider, for example, the Büchi structure $\mathcal{I}$ from Fig. 1 and the Fair-∃AFMC formula

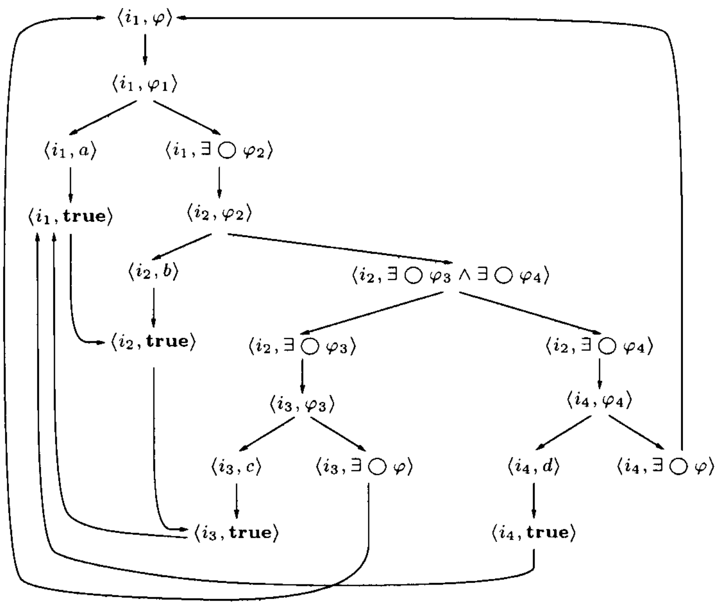
$$\varphi = \nu z.(a \wedge \exists \bigcirc (b \wedge \exists \bigcirc (c \wedge \exists \bigcirc z) \wedge \exists \bigcirc (d \wedge \exists \bigcirc z))).$$

Intuitively, a state of $\mathcal{I}$ belongs to the set defined by the variable z iff it is labeled a and it has a successor labeled b that has two successors, labeled c and d, both having a successor in z. The fact that z is calculated as a greatest fixed-point means that the set of states in z is the largest set that satisfies the above property. In addition, as φ is a Fair-∃AFMC formula, it is required that all infinite runs of $\mathcal{I}$ that are contained in z are fair. A sat-tree of $\mathcal{I}$ for φ is presented in Fig. 4 (in the figure, we use the following abbreviations for formulas in $cl(\varphi)$: $\varphi_4 = d \wedge \exists \bigcirc \varphi$; $\varphi_3 = c \wedge \exists \bigcirc \varphi$; $\varphi_2 = b \wedge \exists \bigcirc \varphi_3 \wedge \exists \bigcirc \varphi_4$; and $\varphi_1 = a \wedge \exists \bigcirc \varphi_2$).

Consider a sat-tree $\langle t, \lambda\rangle$ of $\mathcal{K}$ for ψ. If $\langle t, \lambda\rangle$ contains no node labeled $\langle w, \mathbf{false}\rangle$, then it provides a witness to the satisfaction of all local obligations induced by ψ. In addition, we have to make sure that least fixed-point obligations are not propagated forever and that greatest fixed-point, which are propagated forever along infinite runs, are satisfied along fair runs of $\mathcal{K}$. Formally, the sat-tree $\langle t, \lambda\rangle$ of $\mathcal{K}$ for ψ is *convincing* if the following three conditions hold:

1. The sat-tree $\langle t, \lambda\rangle$ contains no node labeled $\langle w, \mathbf{false}\rangle$. Thus, all local obligations induced by ψ are satisfied.
2. For all infinite paths ρ of $\langle t, \lambda\rangle$, the projection of $\lambda(\rho)$ on the $cl(\psi)$-component contains only finitely many occurrences of formulas of the form $\mu y.f(y)$. Thus, no least fixed-point obligations are propagated forever.
3. For all infinite paths ρ of $\langle t, \lambda\rangle$, the projection of $\lambda(\rho)$ on the W-component satisfies the fairness constraint F of $\mathcal{K}$. Thus, all greatest fixed-point obligations are satisfied along fair runs.

Then, the fair structure $\mathcal{K}$ *satisfies* the Fair-∃AFMC formula ψ, written $\mathcal{K} \models \psi$, if there exists a convincing sat-tree of $\mathcal{K}$ for ψ. For example, the sat-tree from Fig. 4 is convincing. This is because it contains no nodes labeled $\langle w, \mathbf{false}\rangle$ and because its two cycles are permissible. Hence, $\mathcal{I} \models \varphi$.



$\langle i_1, \varphi\rangle$
$\langle i_1, \varphi_1\rangle$
$\langle i_1, a\rangle$
$\langle i_1, \exists\bigcirc\varphi_2\rangle$
$\langle i_1, \mathbf{true}\rangle$
$\langle i_2, \varphi_2\rangle$
$\langle i_2, b\rangle$
$\langle i_2, \exists\bigcirc\varphi_3 \wedge \exists\bigcirc\varphi_4\rangle$
$\langle i_2, \mathbf{true}\rangle$
$\langle i_2, \exists\bigcirc\varphi_3\rangle$
$\langle i_2, \exists\bigcirc\varphi_4\rangle$
$\langle i_3, \varphi_3\rangle$
$\langle i_4, \varphi_4\rangle$
$\langle i_3, c\rangle$
$\langle i_3, \exists\bigcirc\varphi\rangle$
$\langle i_4, d\rangle$
$\langle i_4, \exists\bigcirc\varphi\rangle$
$\langle i_3, \mathbf{true}\rangle$
$\langle i_4, \mathbf{true}\rangle$

FIG. 4. A sat-tree of the Büchi structure $\mathcal{I}$ from Fig. 1 for φ.

Our definition of Fair-∃AFMC is very similar to the automata-theoretic characterization of the alternation-free μ-calculus. Indeed, a convincing sat-tree of $\mathcal{K}$ for ψ can be viewed as a run of an alternating tree automaton for ψ on $\mathcal{K}$ [20]. We also note that for the Fair-∃AFMC formulas that correspond to the existential fragment of Fair-CTL , our definition coincides with the usual semantics for Fair-CTL [6].

Fair Simulation and the Fair-∃AFMC

Before we show that fair simulation and Fair-∃AFMC induce the same relation on fair structures, we demonstrate that this is not the case for ∃-simulation. Consider again the Büchi structures $\mathcal{I}$ and $\mathcal{S}$ from Fig. 1. We saw that the Fair-∃AFMC formula φ is satisfied in $\mathcal{I}$. On the other hand, it is easy to check that although $\mathcal{S}$ ∃-simulates $\mathcal{I}$, the formula φ is not satisfied in $\mathcal{S}$.

THEOREM 5.1. *For all fair structures $\mathcal{K}_1$ and $\mathcal{K}_2$, the following are equivalent*:

(1) *$\mathcal{K}_2$ fairly simulates $\mathcal{K}_1$.*

(2) *For every formula ψ of Fair-∃AFMC, if $\mathcal{K}_1 \models \psi$ then $\mathcal{K}_2 \models \psi$.*

Proof. Assume first that $\mathcal{K}_2$ fairly simulates $\mathcal{K}_1$, and $\mathcal{K}_1 \models \psi$. Then, there exists a convincing sat-tree $\langle t, \lambda\rangle$ of $\mathcal{K}_1$ for ψ. Let $\langle t, \lambda'\rangle$ be the fair trace-tree of $\mathcal{K}_1$ induced by $\langle t, \lambda\rangle$. By Proposition 2.4, this trace-tree is also a fair trace-tree of $\mathcal{K}_2$. Thus, there exists a fair run-tree of $\mathcal{K}_2$ that witnesses $\langle t, \lambda'\rangle$ and which can be used to construct a convincing sat-tree of $\mathcal{K}_2$ for ψ. It follows that $\mathcal{K}_2 \models \psi$.

Assume now that $\mathcal{K}_2$ does not fairly simulate $\mathcal{K}_1$. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be tree automata that accept all fair trace-trees of $\mathcal{K}_1$ and $\mathcal{K}_2$, respectively. By Proposition 2.4, the language of $\mathcal{A}_1$ is not contained in the language of $\mathcal{A}_2$. Hence, by [28], there is a *regular* tree (i.e., a tree with only finitely many distinct subtrees) that is accepted by $\mathcal{A}_1$ and not accepted by $\mathcal{A}_2$. This tree can be encoded by a Fair-∃AFMC formula ψ such that $\mathcal{K}_1 \models \psi$ and $\mathcal{K}_2 \not\models \psi$. ■

Note that when $\mathcal{K}_2$ does not fairly simulate $\mathcal{K}_1$, but $\mathcal{K}_2$ ∃-simulates $\mathcal{K}_1$, there is no Fair-∃-CTL* formula ψ such that $\mathcal{K}_1 \models \psi$ and $\mathcal{K}_2 \not\models \psi$. Thus, the use of fixed-point modalities is essential in this case.

Fair Similarity Quotients

Suppose we are given a fair structure $\mathcal{K}$. We are interested in finding a fair structure $\mathcal{K}'$ that fairly simulates $\mathcal{K}$ and is smaller than $\mathcal{K}$. Due to Theorem 5.1, for every Fair-∀AFMC formula ψ (and in particular, for every Fair-∀CTL formula), if $\mathcal{K}' \models \psi$, then $\mathcal{K} \models \psi$ as well. Thus $\mathcal{K}'$ is a property-preserving abstraction of $\mathcal{K}$, for properties expressed in Fair-∀AFMC.

Let $K = \langle \Sigma, W, \hat{w}, R, L \rangle$ be a structure, and let $\mathcal{K} = \langle K, F \rangle$ be a fair structure with a Büchi fairness constraint F. Note that $\mathcal{K}$ fairly simulates itself. Let $S \subseteq W \times W$ be a maximal fair simulation of $\mathcal{K}$ by $\mathcal{K}$; i.e., no superset of S is a fair simulation. Define a binary relation $E \subseteq W \times W$ as follows: $E(w, w')$ iff $S(w, w')$ and $S(w', w)$. It is easy to see that E is reflexive, symmetric, and transitive. Thus, E is an equivalence relation. The structure $K' = \langle \Sigma, W', \hat{w}', R', L' \rangle$ is defined as follows:

- State set: $W' = W/E$, the equivalence classes of W with respect to E. We denote the equivalence class of state $w \in W$ by $[w]$.
- Initial state: $\hat{w}' = [\hat{w}]$.
- Transition relation: $R' = \{([w], [w']) \mid R(w, w')\}$.
- Labeling function: $L'([w]) = L(w)$. Note that L' is well defined.

The fair structure $\mathcal{K}' = \langle K', F' \rangle$, where $F' = \{[w] \mid [w] \cap F \neq \{\}\}$, is called the *fair-similarity quotient* of $\mathcal{K}$.

PROPOSITION 5.1. *For every Büchi structure $\mathcal{K}$, the fair-similarity quotient $\mathcal{K}'$ of $\mathcal{K}$ fairly simulates $\mathcal{K}$.*

Proof. Consider the relation $S \subseteq W \times W'$, given by $S = \{(w, [w]) \mid w \in W\}$. Let τ be the strategy that maps $(\langle w_0, [w_0] \rangle, \langle w_1, [w_1] \rangle, \ldots, s)$, where $w_0 \cdot w_1 \cdots s$ is a run of K, to $[s]$. Then, S is a fair simulation of $\mathcal{K}$ by $\mathcal{K}'$, with τ as the witnessing strategy. ■

We do not know if the fairness constraints constructed here are minimal. Also, constructing fair-similarity quotients of Streett structures remains open.

6. CONCLUSIONS

We presented a novel extension of the simulation preorder for labeled transition systems to account for fairness. Our definition enjoys a fully abstract tree semantics and has a logical characterization: $\mathcal{S}$ fairly simulates $\mathcal{I}$ iff every fair computation tree embedded in the unrolling of $\mathcal{I}$ can be embedded also in the unrolling of $\mathcal{S}$ or, equivalently, iff every Fair-∀AFMC formula satisfied by $\mathcal{S}$ is satisfied also by $\mathcal{I}$. The locality of the definition leads to a polynomial-time algorithm for checking fair simulation for finite-state systems with weak and strong fairness constraints. Investigating the properties of fair simulation, we related fair simulation to the definitions that already exist in the literature (∃-simulation and ∀-simulation) and argued in favor of our definition. In another paper [14], we show that fair simulation can be checked in a compositional framework, using assume-guarantee reasoning. In [15], we studied a notion of *fair bisimulation*, which is based on a similar idea as the presented notion of fair simulation.

ACKNOWLEDGMENTS

We thank Ramin Hojati, Doron Bustan, and the anonymous reviewers for their comments on this paper.

REFERENCES

1. Abadi, M., and Lamport, L. (1991), The existence of refinement mappings, *Theoret. Comput. Sci.* **82**, 253–284.
2. Aziz, A., Singhal, V., Balarin, F., Brayton, R. K., and Sangiovanni-Vincentelli, A. L. (1994), Equivalences for fair Kripke structures, *in* "ICALP 94: International Colloquium on Automata, Languages, and Programming," Lecture Notes in Computer Science, Vol. 820, pp. 364–375, Springer-Verlag, Berlin/New York.
3. Bensalem, S., Bouajjani, A., Loiseaux, C., and Sifakis, J. (1992), Property-preserving simulations, *in* "CAV 92: Computer Aided Verification," Lecture Notes in Computer Science, Vol. 663, pp. 260–273, Springer-Verlag, Berlin/New York.
4. Bergstra, J. A., Klop, J. W., and Olderog, E. R. (1987), Failures without chaos: A new process semantics for fair abstraction, *in* "Formal Description Techniques III," proc. third IFIP WG 2.2 working cont. (M. Wirsing ed.), Ebberup, Denmark 1986, North Holland, pp. 77–103.

5. Bloom, B., and Paige, R. (1995), Transformational design and implementation of a new efficient solution to the ready simulation problem, *Sci. Comput. Programming* **24**, 189–220.
6. Clarke, E. M., Emerson, E. A., and Sistla, A. P. (1986), Automatic verification of finite-state concurrent systems using temporal-logic specifications, *ACM Trans. Programming Languages Systems* **8**, 244–263.
7. Cleaveland, R., and Natarajan, V. (1995), Divergence and fair testing, *in* "ICALP '95: International Colloquium on Automata, Languages, and Programming," Lecture Notes in Computer Science, Vol. 944, pp. 648–659, Springer-Verlag, Berlin/New York.
8. Cleaveland, R. J., Parrow, J., and Steffen, B. (1993), The concurrency workbench: A semantics-based tool for the verification of finite-state systems, *ACM Trans. Programming Languages Systems* **15**, 36–72.
9. Damm, W., and Pnueli, A. (1997), Verifying out-of-order executions, *in* "CHARME 97: Correct Hardware Design and Verification Methods," pp. 23–47.
10. Dill, D. L., Hu, A. J., and Wong-Toi, H. (1991), Checking for language inclusion using simulation relations, *in* "CAV 91: Computer Aided Verification," Lecture Notes in Computer Science, Vol. 575, pp. 255–265, Springer-Verlag, Berlin/New York.
11. Grumberg, O., and Long, D. E. (1994), Model checking and modular verification, *ACM Trans. Programming Languages Systems* **16**, 843–871.
12. Hennessy, M. C. B. (1987), An algebraic theory of fair asynchronous communicating processes, *Theoret. Comput. Sci.* **49**, 121–143.
13. Henzinger, M. R., Henzinger, T. A., and Kopke, P. W. (1995), Computing simulations on finite and infinite graphs, *in* "FOCS 95: Foundations of Computer Science," pp. 453–462.
14. Henzinger, T. A., Qadeer, S., Rajamani, S. K., and Tasiran, S. (1998), An assume–guarantee rule for checking simulation, *in* "FMCAD 98: Formal Methods in Computer-Aided Design," Lecture Notes in Computer Science, Vol. 1522, pp. 421–432, Springer-Verlag, Berlin/New York.
15. Henzinger, T. A., and Rajamani, S. K. (2000), Fair bisimulation, *in* "TACAS 00: Tools and Algorithms for Construction and Analysis of Systems," Lecture Notes in Computer Science, Vol. 1785, pp. 299–314, Springer-Verlag, Berlin/New York.
16. Hojati, R. (1996), "A BDD-Based Environment for Formal Verification of Hardware Systems," Ph.D. thesis, University of California, Berkeley.
17. Kupferman, O., Safra, S., and Vardi, M. Y. (1996), Relating word and tree automata, *in* "LICS 96: Logic in Computer Science," pp. 322–333.
18. Kupferman, O., and Vardi, M. Y. (1996), Verification of fair transition systems, *in* "CAV 96: Computer Aided Verification," Lecture Notes in Computer Science, Vol. 1102, pp. 372–381, Springer-Verlag, Berlin/New York.
19. Kupferman, O., and Vardi, M. Y. (1998), Weak alternating automata and tree automata emptiness, *in* "STOC 98: Symposium on Theory of Computing," pp. 224–233.
20. Kupferman, O., Vardi, M. Y., and Wolper, P. (2000), An automata theoretic approach to branching time model checking, *J. Assos. Comput. Mach.* **47**, 312–360.
21. Lamport, L. (1983), Specifying concurrent program modules, *ACM Trans. Programming Languages Systems* **5**, 190–222.
22. Lynch, N. A., and Segala, R. (1993), "A Comparison of Simulation Techniques and Algebraic Techniques for Verifying Concurrent Systems," Technical Report, MIT/LCS/TM-499, MIT, Cambridge, MA.
23. Lynch, N. A., and Tuttle, M. R. (1987), Hierarchical correctness proofs for distributed algorithms, *in* "Proceedings of the 6th Annual Symposium on Principles of Distributed Computing," pp. 137–151.
24. Lynch, N. A. (1996), "Distributed Algorithms," Morgan Kaufmann, San Mateo, CA.
25. Milner, R. (1971), An algebraic definition of simulation between programs, *in* "Proceedings of the 2nd International Joint Conference on Artificial Intelligence," pp. 481–489.
26. Pnueli, A. (1985), Linear and branching structures in the semantics and logics of reactive systems, *in* "ICALP 85: International Colloquium on Automata, Languages, and Programming," Lecture Notes in Computer Science, Vol. 194, pp. 15–32, Springer-Verlag, Berlin/New York.
27. Pnueli, A., and Rosner, R. (1989), On the synthesis of a reactive module, *in* "POPL 89: Principles of Programming Languages," pp. 179–190.
28. Rabin, M. O. (1970), Weakly definable relations and special automata, *in* "Proceedings of the Symposium on Mathematical Logic and Foundations of Set Theory," pp. 1–23.
29. Safra, S. (1988), On the complexity of ω-automata, *in* "FOCS 88: Foundations of Computer Science," pp. 319–327.
30. Sistla, A. P., Vardi, M. Y., and Wolper, P. (1987), The complementation problem for Büchi automata with applications to temporal logic, *Theoret. Comput. Sci.* **49**, 217–237.
31. Stockmeyer, L. J., and Meyer, A. R. (1973), Word problems requiring exponential time, *in* "STOC 73: Symposium on Theory of Computing," pp. 1–9.
32. Vogler, W., Brinksma, E., and Rensink, A. (1995), Fair testing, *in* "CONCUR 95: Theories of Concurrency," Lecture Notes in Computer Science, Vol. 962, pp. 313–327, Springer-Verlag, Berlin/New York.