Chicago Journal of Theoretical Computer Science

The MIT Press

Volume 1998, Article 2
16 March 1998

ISSN 1073–0486. MIT Press Journals, Five Cambridge Center, Cambridge, MA 02142-1493 USA; (617)253-2889; *journals-orders@mit.edu, journals-info@mit.edu*. Published one article at a time in LaTeX source form on the Internet. Pagination varies from copy to copy. For more information and other articles see:

- *http://mitpress.mit.edu/CJTCS/*
- *http://www.cs.uchicago.edu/publications/cjtcs/*
- *ftp://mitpress.mit.edu/pub/CJTCS*
- *ftp://cs.uchicago.edu/pub/publications/cjtcs*

The *Chicago Journal of Theoretical Computer Science* is abstracted or indexed in *Research Alert,®* *SciSearch,®* *Current Contents®/Engineering Computing & Technology*, and *CompuMath Citation Index.®*

The *Chicago Journal of Theoretical Computer Science* is a peer-reviewed scholarly journal in theoretical computer science. The journal is committed to providing a forum for significant results on theoretical aspects of all topics in computer science.

Verification of Fair Transition Systems

Orna Kupferman Moshe Y. Vardi

16 March, 1998

Abstract

Abstract-1 In program verification, we check that an implementation meets its specification. Both the specification and the implementation describe the possible behaviors of the program, although at different levels of abstraction. We distinguish between two approaches to implementation of specifications. The first approach is trace-based implementation, where we require every computation of the implementation to correlate to some computation of the specification. The second approach is tree-based implementation, where we require every computation tree embodied in the implementation to correlate to some computation tree embodied in the specification. The two approaches to implementation are strongly related to the linear-time versus branching-time dichotomy in temporal logic.

Abstract-2 In this work, we examine the trace-based and the tree-based approaches from a complexity-theoretic point of view. We consider and compare the complexity of verification of fair transition systems, modeling both the implementation and the specification in the two approaches. We consider unconditional, weak, and strong fairnesses. For the trace-based approach, the corresponding problem is fair containment. For the tree-based approach, the corresponding problem is fair simulation. We show that while both problems are PSPACE-complete, their complexities in terms of the size of the implementation do not coincide, and the trace-based approach is easier. As the implementation is normally much bigger than the specification, we see this as an advantage of the trace-based approach. Our results are at variance with the known results for the case of transition systems with no fairness, where no approach is evidently advantageous.

1 Introduction

1-1 In *program verification*, we check that an *implementation* meets its *specification*. Both the specification and the implementation describe the possible behaviors of the program, but the implementation is more concrete than the specification, or, equivalently, the specification is more abstract than the implementation [AL91]. This basic notion of verification suggests a top-down method for design development. Starting with a highly abstract specification, we can construct a sequence of "behavior descriptions." Each description refers to its predecessor as a specification, so it is less abstract than its predecessor. The last description contains no abstractions, and constitutes the implementation. Hence the name *hierarchical refinement* for this methodology [LS84, LT87, Kur94].

1-2 We distinguish between two approaches to implementation of specifications. The first approach is *trace-based implementation*, where we require every computation of the implementation to correlate to some computation of the specification. The second approach is *tree-based implementation*, where we require every computation tree embodied in the implementation to correlate to some computation tree embodied in the specification. The exact notion of correct implementation then depends on how we interpret correlation. We can, for example, interpret correlation as identity. Then, a correct trace-based implementation is one in which every computation is also a computation of the specification, and a correct tree-based implementation is one in which every embodied computation tree is also embodied in the specification. Numerous interpretations of correlation are suggested and studied in the literature [Hen85, Mil89, AL91]. Here we consider a simple definition of correlation, and interpret it as equivalence with respect to the variables joint to the implementation and the specification, as the implementation is typically defined over a wider set of variables (reflecting the fact that it is more concrete than the specification).

1-3 The tree-based approach is stronger in the following sense. If $\mathcal{I}$ is a correct tree-based implementation of the specification $\mathcal{S}$, then $\mathcal{I}$ is also a correct trace-based implementation of $\mathcal{S}$. As shown by Milner [Mil80], the opposite situation is not true. The two approaches to implementation are strongly related to the linear-time versus branching-time dichotomy in temporal logic [Pnu85]. The temporal-logic analogy to the strength of the tree-based approach is the expressiveness superiority of $\forall$CTL*, the universal fragment of CTL*, over LTL [CD88]. Indeed, while a correct trace-based implementation is guaranteed to satisfy all the LTL formulas satisfied in the specification, a correct tree-based implementation is guaranteed to satisfy all the $\forall$CTL* formulas satisfied in the specification [GL94].

1-4 In this work, we examine the traced-based and tree-based approaches from a complexity-theoretic point of view. More precisely, we consider and compare the complexity of the problem of determining whether $\mathcal{I}$ is a correct trace-based implementation of $\mathcal{S}$, and the problem of determining whether $\mathcal{I}$ is a correct tree-based implementation of $\mathcal{S}$. The different levels of abstraction in the implementation and the specification are reflected in their sizes. The implementation is typically much larger than the specification, and it is the implementation's size that is the computational bottleneck. Therefore, of particular interest to us is the *implementation complexity* of these problems; i.e., their complexity in terms of $\mathcal{I}$, assuming $\mathcal{S}$ is fixed.[1]

1-5 We model specifications and implementations by *transition systems* [Kel76]. The systems are defined over the sets $AP_{\mathcal{I}}$ and $AP_{\mathcal{S}}$ of *atomic propositions* used in the implementation and specification, respectively. Thus, the alphabets of the systems are $2^{AP_{\mathcal{I}}}$ and $2^{AP_{\mathcal{S}}}$. Recall that usually the implementation has more variables than the specification; hence, $AP_{\mathcal{I}} \supseteq AP_{\mathcal{S}}$. We therefore interpret correlation as equivalence with respect to $AP_{\mathcal{S}}$. In other words, associating behaviors of the implementation with those of the specification, we first project the behaviors onto $AP_{\mathcal{S}}$. Within this framework, correct trace-based implementation corresponds to *containment*, and correct tree-based implementation corresponds to *simulation* [Mil71].

1-6 We start by reviewing and examining transition systems with no fairness conditions. It is well known that simulation can be checked in polynomial time [Mil80, BGS92, AV95, HHK95], whereas the containment problem is in PSPACE [SVW87]. We show that the latter problem is PSPACE-hard; thus the tree-based approach is easier than the trace-based approach. Yet, once we turn to consider the implementation complexity of simulation and containment, the trace-based approach is easier than the tree-based approach. Indeed, we show that the implementation complexity of simulation is PTIME-complete, which is most likely harder than the NLOGSPACE-complete bound for the implementation complexity of containment. For

[1]The distinction between the relative sizes of the implementation and the specification is less significant in the context of hierarchical refinement. There, each internal behavior description is both an implementation (of some step i in the chain of successive refinements) and a specification (of step $i+1$ in the chain). Thus, the distinction is crucial only in the last refinement step, which handles the final, and hence largest, implementation.

these reasons, when we consider transition systems with no fairness, there is no clear advantageous approach. This is reminiscent of the computational relations of branching-time model checking and linear-time model checking: while model checking is easier for the branching paradigm, the implementation complexity of model checking in the two paradigms coincide [LP85, CES86, VW86, BVW94].

1-7 Often, we want our implementations and specifications to describe behaviors that satisfy both liveness and safety properties. Then, transition systems with no fairness condition are too weak, and we need the framework of *fair transition systems*. We consider unconditional, weak, and strong fairness (also known as impartiality, justice, and compassion, respectively) [LPS81, Eme90, MP92]. Within this framework, correct trace-based implementation corresponds to *fair containment*, and correct tree-based implementation corresponds to *fair simulation* [BBLS92, ASB$^+$94, GL94]. Hence, it is the complexity of these problems that should be examined when we compare the trace-based and the tree-based approaches.

1-8 We present a uniform method and a simple algorithm for solving the fair-containment problem for all three types of fairness conditions. Unlike [CDK93], we consider the case where both the specification and the implementation are nondeterministic, as is appropriate in a hierarchical refinement framework. We prove that the problem is PSPACE-complete for all three types. For the case where the implementation uses the unconditional or weak fairness conditions, our nondeterministic algorithm requires space logarithmic in the size of the implementation (regardless of the type of fairness condition used in the specification). For the case where the implementation uses the strong fairness condition, we suggest an alternative algorithm that runs in time polynomial in the size of the implementation. We show that these algorithms are optimal; the implementation complexity of fair containment is NLOGSPACE-complete for implementations that use the unconditional or weak fairness conditions [VW94] and is PTIME-complete for implementations that use the strong fairness condition. To prove the latter, we show that the nonemptiness problem for fair transition systems with a strong fairness condition is PTIME-hard.

1-9 We also present a uniform method and a simple algorithm for solving the fair-simulation problem for the three types of fairness conditions. Our algorithm uses the fair-containment algorithm as a subroutine. We prove that the problem is PSPACE-complete for all three types. Like Milner's algorithm for checking simulation [Mil90], our algorithm can be implemented as a calculation of a fixed-point expression. The running time of our algorithm is polynomial in the size of the implementation. We show that this is optimal; thus, the implementation complexity of fair simulation is PTIME-complete for all types of fairness conditions. In fact, in proving the latter, we prove that the implementation complexity of simulation (without fairness conditions) is PTIME-hard as well.

1-10 Our results (summarized in Table 1) show that when we model the specification and the implementation by fair transition systems, or we consider the implementation complexity of verification, the computational advantage of the tree-based approach disappears. Furthermore, when we consider the implementation complexity, checking implementations that use unconditional or weak fairness conditions is easier in the trace-based approach.

2 Preliminaries

2.1 Fair Transition Systems

2.1-1 A fair transition system (transition system, for short) $S = \langle \Sigma, W, R, W_0, L, \alpha \rangle$ consists of an alphabet Σ, a finite set W of states, a total transition relation $R \subseteq W \times W$ (i.e., for every $w \in W$ there exists $w' \in W$ such that $R(w, w')$), a set W_0 of initial states, a labeling function $L : W \to \Sigma$, and a fairness condition α. We will define three types of fairness conditions shortly. A computation of S is a sequence $\pi = w_0, w_1, w_2, \ldots$ of states such that for every $i \geq 0$, we have $R(w_i, w_{i+1})$. For an alphabet Σ, we use Σ^* and Σ^ω to denote the sets of all finite and infinite words over Σ, respectively. For two words $\rho_1 \in \Sigma^*$ and $\rho_2 \in \Sigma^* \cup \Sigma^\omega$, we use $\rho_1 \cdot \rho_2$ to denote the concatenation of ρ_1 and ρ_2. Each computation $\pi = w_0, w_1, w_2, \ldots$ induces the word $L(\pi) = L(w_0) \cdot L(w_1) \cdot L(w_2) \ldots \in \Sigma^\omega$.

2.1-2 To determine whether a computation is fair, we refer to the set $Inf(\pi)$ of states that π visits infinitely often. Formally,

$$Inf(\pi) = \{w \in W \ :\ \text{for infinitely many } i \geq 0, \text{ we have } w_i = w\}$$

The way we refer to $Inf(\pi)$ depends upon the fairness condition of S. Several types of fairness conditions are studied in the literature. Here we consider three:

- Unconditional fairness (or impartiality), where $\alpha \subseteq W$, and π is fair if and only if $Inf(\pi) \cap \alpha \neq \emptyset$.

- Weak fairness (or justice), where $\alpha \subseteq 2^W \times 2^W$, and π is fair if and only if for every pair $\langle B, G\rangle \in \alpha$, we have that $Inf(\pi) \cap (W \setminus B) = \emptyset$ implies $Inf(\pi) \cap G \neq \emptyset$.

- Strong fairness (or fairness), where $\alpha \subseteq 2^W \times 2^W$, and π is fair if and only if for every pair $\langle B, G\rangle \in \alpha$, we have that $Inf(\pi) \cap B \neq \emptyset$ implies $Inf(r) \cap G \neq \emptyset$.

2.1-3 In addition, we consider *nonfair transition systems*; i.e., transition systems in which all the computations are fair.

2.1-4 It is easy to see that fair transition systems are essentially notational variants of automata on infinite words [Tho90]. In particular, the unconditional and the strong fairness conditions correspond to the Büchi and Streett acceptance conditions. This correspondence motivates our noncanonical definition of fairness, where the unconditional fairness condition consists of a single constraint, whereas the weak and strong fairness conditions consist of a conjunction of constraints. In the sequel, we use results from the theory of Büchi and Streett automata in the context of fair transition systems.

2.1-5 Defining the unconditional fairness condition as a conjunction of constraints (that is, having $\alpha \subseteq 2^W$) corresponds to the generalized Büchi condition from automata theory. Note that such generalized unconditional fairness conditions are very similar to weak fairness conditions; in fact, they can be viewed as a special case of weak fairness conditions, with $B = W$ in all pairs. Moreover, as we state formally in Lemma 1, weak fairness conditions can be easily translated to generalized unconditional fairness conditions. Indeed, a pair $\langle B, G\rangle$ in a weak fairness condition is equivalent to a set $(W \setminus B) \cup G$ in a generalized unconditional fairness condition.

2.1-6 For a state w, a w-computation is a computation $w_0, w_1, w_2, \ldots$ with $w_0 = w$. We use $\mathcal{T}(S^w)$ to denote the set of all traces $\sigma_0 \cdot \sigma_1 \ldots \in \Sigma^\omega$ for which there exists a fair w-computation $w_0, w_1, \ldots$ in S with $L(w_i) = \sigma_i$ for all $i \geq 0$. The *trace set* $\mathcal{T}(S)$ of S is then defined as $\bigcup_{w \in W_0} \mathcal{T}(S^w)$. Thus, each transition system defines a subset of Σ^ω. We say that a transition system S is *empty* if and only if $\mathcal{T}(S) = \emptyset$; i.e., S has no fair computation. We sometimes say that S accepts a trace ρ, meaning that $\rho \in \mathcal{T}(S)$. Note that for a nonfair transition system S, the set $\mathcal{T}(S)$ contains all traces $\rho \in \Sigma^\omega$ for which there exists a computation π with $L(\pi) = \rho$.

2.1-7 The size of a transition system and its fairness condition determine the complexity of solving questions about it. We define classes of transition systems according to these two characteristics. We use $\mathcal{U}, \mathcal{W}$, and $\mathcal{S}$ to distinguish between the unconditional, weak, and strong fairness conditions, respectively. We measure the size of a transition system by the number of its states (the number of edges is at most quadratic in the number of states) and, in the case of weak and strong fairness, also by the number of pairs in its fairness condition. For example, the set of unconditionally fair transition systems with n states is denoted $\mathcal{U}(n)$. We also use a line over the transition system to denote the complementary transition system (one that accepts the complementary trace set). For example, the set of transition systems that complements strongly fair transition systems with n states and m pairs is denoted by $\overline{\mathcal{S}(n,m)}$. By [Saf88, Saf92], every fair transition system indeed has a complementary transition system. We use $\rightarrow$ to denote a possible translation (that preserves the trace sets of the systems) between various types of transition systems. For example, $\mathcal{W}(n,m) \rightarrow \mathcal{U}(nm)$ means that each weakly fair transition system S with n states and m pairs can be translated to an unconditionally fair transition system S' with nm states such that $\mathcal{T}(S) = \mathcal{T}(S')$.

2.2 Trace-Based and Tree-Based Implementations

2.2-1 In this section, we formalize what it means for an implementation S to correctly implement a specification S', in both the trace-based and the tree-based approaches. Recall that S and S' are given as fair transition systems over the alphabets 2^{AP} and $2^{AP'}$, respectively, with $AP \supseteq AP'$. For technical convenience, we assume that $AP = AP'$; thus, the implementation and the specification are defined over the same alphabet. By taking, for each $\sigma \in 2^{AP}$, the letter $\sigma \cap AP'$ instead of the letter σ, all of our algorithms and results are also valid for the case $AP \supset AP'$.

2.2-2 Recall that nonfair transition systems are of special interest. The problems that formalize correct trace-based and tree-based implementations in this framework are containment and simulation. Once we add fairness to the systems, the corresponding problems are fair containment and fair simulation. Below we define the four problems. All problems are defined with respect to two transition systems: $S = \langle \Sigma, W, R, W_0, L, \alpha \rangle$, and $S' = \langle \Sigma, W', R', W'_0, L', \alpha' \rangle$.

2.2.1 Containment and Fair Containment

2.2.1-1 The fair-containment problem of S and S' is to determine whether $\mathcal{T}(S) \subseteq \mathcal{T}(S')$; that is, whether every trace accepted by S is also accepted by S'. When S and S' are nonfair, we call the problem containment.

2.2.1-2 While containment and fair containment refer only to the set of computations of S and S', simulation and fair simulation also refer to the branching structure of the transition systems.

2.2.2 Simulation

2.2.2-1 Let w and w' be states in W and W', respectively. A relation $H \subseteq W \times W'$ is a simulation relation from $\langle S, w\rangle$ to $\langle S', w'\rangle$ if and only if the following conditions hold [Mil71]:

1. $H(w, w')$.
2. For all t and t' with $H(t, t')$, we have $L(t) = L(t')$.
3. For all t and t' with $H(t, t')$ and for all $s \in W$ such that $R(t, s)$, there exists $s' \in W'$ such that $R'(t', s')$ and $H(s, s')$.

2.2.2-2 A simulation relation H is a simulation from S to S' if and only if for every $w \in W_0$ there exists $w' \in W_0'$ such that $H(w, w')$. If there exists a simulation from S to S', we say that S simulates S', and we write $S \leq S'$. Intuitively, this means that the transition system S' has more behaviors than the transition system S. In fact, every $\forall$CTL* formula that is satisfied in S' is satisfied also in S [BCG88]. Given S and S', the simulation problem is to determine whether $S \leq S'$.

2.2.2-3 We also mention here the *bisimulation problem* [Mil71]. Two transition systems are bisimilar if and only if they have exactly the same behavior. Formally, a relation $H \subseteq W \times W'$ from $\langle S, w\rangle$ to $\langle S', w'\rangle$ is a bisimulation if, in addition to conditions 1–3 above, the following condition also holds:

4. For all t and t' with $H(t, t')$ and for all $s' \in W'$ such that $R'(t', s')$, there exists $s \in W$ such that $R(t, s)$ and $H(s, s')$.

2.2.3 Fair Simulation

2.2.3-1 Let $H \subseteq W \times W'$ be a relation over the states of S and S'. It is convenient to extend H to also relate infinite computations of S and S'. For two computations $\pi = w_0, w_1, \ldots$ in S, and $\pi' = w_0', w_1', \ldots$ in S', we say that $H(\pi, \pi')$

holds if and only if $H(w_i, w'_i)$ holds for all $i \geq 0$. For a pair $\langle w, w' \rangle \in W \times W'$, we say that $\langle w, w' \rangle$ is *good in* H if and only if for every fair w-computation π in S, there exists a fair w'-computation π' in S', such that $H(\pi, \pi')$.

2.2.3-2 Let w and w' be states in W and W', respectively. A relation $H \subseteq W \times W'$ is a fair-simulation relation from $\langle S, w \rangle$ to $\langle S', w' \rangle$ if and only if the following conditions hold [GL94]:

1. $H(w, w')$.
2. For all t and t' with $H(t, t')$, we have $L(t) = L(t')$.
3. For all t and t' with $H(t, t')$, the pair $\langle t, t' \rangle$ is good in H.

2.2.3-3 A fair-simulation relation H is a fair simulation from S to S' if and only if for every $w \in W_0$ there exists $w' \in W'_0$ such that $H(w, w')$. If there exists a fair simulation from S to S', we say that S *fair simulates* S', and we write $S \leq S'$. Intuitively, this means that the transition system S' has more fair behaviors than the transition system S. In fact, every fair-$\forall$CTL* formula (that is, every $\forall$CTL* formula with path quantification ranging only over fair computations [EL87]) that is satisfied in S' is also satisfied in S [GL94]. The fair-simulation problem is, given S and S', to determine whether $S \leq S'$. Note that when they relate nonfair transition systems, fair simulation and simulation coincide.

2.2.3-4 We also mention here the *fair-bisimulation problem* [GL94]. Two transition systems are bisimilar if and only if they have exactly the same fair behavior. Formally, a relation $H \subseteq W \times W'$ from $\langle S, w \rangle$ to $\langle S', w' \rangle$ is a fair bisimulation if conditions 1–3 hold with the following symmetric definition of when a pair is good in H. In fair bisimulation, a pair $\langle w, w' \rangle \in W \times W'$ is good in H if and only if for every fair w-computation π in S, there exists a fair w'-computation π' in S' such that $H(\pi, \pi')$, and for every fair w'-computation π' in S', there exists a fair w-computation π in S such that $H(\pi, \pi')$.

2.2.3-5 It is easy to see that simulation implies containment, and that fair simulation implies fair containment; that is, if $S \leq S'$, then $\mathcal{T}(S) \subseteq \mathcal{T}(S')$. The opposite, however, is not true. Consider the two transition systems S and S' presented in Figure 1. Note that while the system S has two initial states, one labeled by a and one labeled by b, the system S' has three initial states, two labeled by a and one labeled by b. The trace sets of both transition systems are $(a+b)^\omega$. As such, $\mathcal{T}(S) \subseteq \mathcal{T}(S')$, but still, S does not simulate S'. Indeed, no initial state of S' can be paired, by any H, to the initial state labeled a of S.

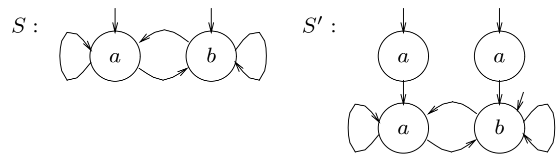

Figure 1: $\mathcal{T}(S) \subseteq \mathcal{T}(S')$ but $S \not\leq S'$

2.3 Complexity Measures

2.3-1 In the rest of this paper, we examine the traced-based and the tree-based approaches from a complexity-theoretic point of view. We consider and compare the complexity of the four problems. The different levels of abstraction in the implementation and the specification are reflected in their sizes. The implementation is typically much larger than the specification, and it is the implementation's size that is the computational bottleneck. Therefore, of particular interest to us is the implementation complexity of these problems; i.e., the complexity of checking whether $\mathcal{T}(S) \subseteq \mathcal{T}(S')$ and $S \leq S'$, in terms of the size of S, assuming S' is fixed.

2.3-2 We refer to three complexity classes: PSPACE, PTIME, and NLOGSPACE, briefly described as follows (for a full definition, see [GJ79]). We say that a problem is *in* PSPACE (PTIME) if there exists a deterministic algorithm for solving the problem such that the working space (time, respectively) required for executing the algorithm is polynomial in the size of its input. The problem is in NLOGSPACE if there exists a nondeterministic algorithm for solving the problem such that the working space required for the algorithm is logarithmic in the size of its input. A program $\mathcal{P}$ is complete for a complexity class C if and only if $\mathcal{P}$ is in C and is also C-hard.

2.3-3 Some of the results we study, mainly those that we consider nonfair transition systems, are known; therefore we only review them.

3 Verification of Nonfair Transition Systems

3-1 In this section, we study the complexity of the containment and simulation problems. We show that while the containment problem is harder, its implementation complexity is lower than that of the simulation problem.

3.1 The Containment Problem

Theorem 1 *The containment problem is PSPACE-complete.*

Proof of Theorem 1-1

Proof of Theorem 1 The upper bound follows from known PSPACE bound to the fair-containment problem for unconditionally fair transition systems ([SVW87], see also Theorem 5).

Proof of Theorem 1-2

We can regard an unconditionally fair system $S = \langle \Sigma, W, R, W_0, L, \alpha \rangle$, with $\alpha \subseteq W$, as a *finite-acceptance* transition system. The traces of a finite-acceptance transition system are finite words over the alphabet Σ. A finite computation $\pi = w_0, \ldots, w_k$ is "fair" in S (that is, $L(\pi)$ is accepted) if and only if $w_k \in \alpha$. We call the states in α *final* states. A finite-acceptance transition system S is universal if and only if $\mathcal{T}(S) = \Sigma^*$. In [MS72], Mayer and Stockmayer prove a PSPACE lower bound for the problem of determining whether a finite-acceptance transition system S is universal (the framework in [MS72] is of regular expressions, yet regular expressions can be linearly translated to finite-acceptance transition systems).

Proof of Theorem 1-3

We reduce the universality problem for finite-acceptance transition systems to the containment problem. Consider a finite-acceptance transition system $S = \langle \Sigma, W, R, W_0, L, \alpha \rangle$. Let

$$S' = \langle \Sigma \cup \{\#\}, W \cup \{w_\#\}, R \cup R_\#, W_0, L', W \cup \{w_\#\} \rangle$$

be a nonfair transition system, where:

- For every $w \in (\alpha \cup \{w_\#\})$, we have $R_\#(w, w_\#)$. Thus S' has the transitions of S augmented with transitions from all the final states to the state $w_\#$, which has a self loop.
- For every state $w \in W$, we have $L'(w) = L(w)$. In addition, $L'(w_\#) = \{\#\}$.

Proof of Theorem 1-4

We prove that $\Sigma^* \subseteq \mathcal{T}(S)$ if and only if $\Sigma^\omega \cup (\Sigma^* \cdot \{\#\}^\omega) \subseteq \mathcal{T}(S')$. Assume first that $\Sigma^\omega \cup (\Sigma^* \cdot \{\#\}^\omega) \subseteq \mathcal{T}(S')$, and assume, by way of contradiction, that there exists $\rho \in \Sigma^*$ such that $\rho \notin \mathcal{T}(S)$. Let $\pi = w_0, w_1, \ldots$ be an accepting computation of S' on $\rho \cdot \#^\omega$. By the definition of L', we have that $\pi \in W^* \cdot w_\#^\omega$. Let k be such that $w_k \neq w_\#$ and $w_{k+1} = w_\#$. As $R_\#(w_k, w_{k+1})$, it must be that $w_k \in \alpha$. In addition, for all $0 \leq i < k-1$, we have $R(w_i, w_{i+1})$. Hence $w_0, \ldots, w_k$ is an accepting computation of S on ρ, and we reach a contradiction.

Proof of Theorem 1-5

Assume now that $\Sigma^* \subseteq \mathcal{T}(S)$. Consider a trace $\rho \in \Sigma^*$. Let $w_0, w_1, \ldots, w_k$ be an accepting computation of S on ρ. Clearly, $w_0, w_1, \ldots, w_k, w_{\#}^{\omega}$ is an accepting computation of S' on $\rho \cdot \{\#\}^{\omega}$. Hence, $\Sigma^* \cdot \{\#\}^{\omega} \subseteq \mathcal{T}(S')$. Next, consider a trace $\rho = \sigma_0, \sigma_1, \ldots \in \Sigma^{\omega}$. We define a tree that embodies all the possible runs of S on finite prefixes of ρ. The tree has a root labeled ϵ. The nodes of level 1 (that is, nodes that have a path of length 1 from the root) are states w_0 in W_0 for which $L(w_0) = \sigma_0$. For $l \geq 0$, a node w of level i has as successors nodes w' for which $R(w, w')$ and $L(w') = \sigma_{l+1}$. Because $\Sigma^* \subseteq \mathcal{T}(S)$, the tree embodies accepting runs for all the (infinitely many) finite prefixes of ρ, and is therefore infinite. Hence, by König's lemma, we can pick a sequence $\pi = \epsilon, w_0, w_1, \ldots$ such that $w_0 \in W_0$, and for all $j \geq 0$, we have that $L(w_j) = \sigma_j$ and $R(w_j, w_{j+1})$. As such, π is an accepting run of S' on ρ. As we can easily construct a nonfair transition system for $\Sigma^{\omega} \cup (\Sigma^* \cdot \{\#\}^{\omega})$, we are done.

Proof of Theorem 1 □

Theorem 2 *The implementation complexity of the containment problem is NLOGSPACE-complete.*

Proof of Theorem 2 In Theorem 6, we prove an NLOGSPACE upper bound for the implementation complexity of the more general fair-containment problem for unconditionally fair transition systems. The lower bound follows easily by a reduction from the nonreachability problem in a directed graph, proved to be NLOGSPACE-complete in [Jon75].[2] Given a directed graph G with two designated nodes s and t, we can view G as a nonfair transition system S_G over $\Sigma = \{\sigma, s, t\}$ in which all states are initial states, all states except s and t are labeled σ, and the states s and t are labeled with s and t, respectively. It is easy to see that the node t is not reachable from s if and only if $\mathcal{T}(G) \subseteq \{\sigma, t\}^* \cdot \{\sigma, s\}^{\omega} \cup \{\sigma, t\}^{\omega}$. Since we can specify the expression in the right with a fixed nonfair transition system, we are done.

Proof of Theorem 2 □

[2]The proof in [Jon75] is for the reachability problem. Yet, for every problem P, we have that P is NLOGSPACE-complete if and only if P is co-NLOGSPACE-complete (see [Imm88, Sze88] for NLOGSPACE = co-NLOGSPACE; the argument for the completeness is easy).

3.2 The Simulation Problem

Theorem 3 *The simulation problem is PTIME-complete.*

Proof of Theorem 3 The upper bound is given in [Mil80]. The lower bound follows from the reduction in [BGS92] (the reduction there proves PTIME-hardness for bisimulation, but it is valid also for simulation). An additional proof of the lower bound is given in Theorem 4.

Proof of Theorem 3 □

Theorem 4 *The implementation complexity of the simulation problem is PTIME-complete.*

Proof of Theorem 4-1 **Proof of Theorem 4** Membership in PTIME follows from Theorem 3.

Proof of Theorem 4-2 We prove hardness in PTIME by reducing the NAND circuit value problem (NANDCV), proved to be PTIME-complete in [Gol77, GHR95], to the problem of determining whether a transition system S simulates a fixed transition system S', both with no fairness. In the NANDCV problem, we are given a Boolean circuit β constructed solely of NAND gates of fanout 2, and a vector $\langle x_1, \ldots, x_n \rangle$ of Boolean input values. The problem is to determine whether the output of β on $\langle x_1, \ldots, x_n \rangle$ is 1. Formally, we denote a Boolean circuit of NAND gates by a tuple $\beta = \langle G, G_{in}, g_{out}, T_l, T_r \rangle$, where G is the set of internal gates, G_{in} is the set of input gates (identified as $g_1, \ldots, g_n$), $g_{out} \in G \cup G_{in}$ is the output gate, $T_l : G \rightarrow G \cup G_{in}$ maps each internal gate to its left input, and $T_r : G \rightarrow G \cup G_{in}$ maps each internal gate to its right input. Note that since there are no circular dependencies in β, the graph induced by T_l and T_r is acyclic.

Proof of Theorem 4-3 The idea of the reduction is as follows. We define a fixed transition system S' that embodies all the NAND circuits β and input vectors $\vec{x}$ for which the value of β on $\vec{x}$ is 1. Then, given a circuit β and an input vector $\vec{x}$, we translate them to a transition system S such that $S \leq S'$ if and only if the value of β on $\vec{x}$ is 1.

Proof of Theorem 4-4 The transition system S' (see Figure 2) has 12 states. Eight states correspond to internal gates. Each of these states is an entry in the truth table of the operator NAND, attributed with a direction, either $\swarrow$ or $\searrow$. Thus, the "internal states" of S' are $\langle 001 \swarrow \rangle$, $\langle 011 \swarrow \rangle$, $\langle 101 \swarrow \rangle$, $\langle 110 \swarrow \rangle$, $\langle 001 \searrow \rangle$, $\langle 011 \searrow \rangle$, $\langle 101 \searrow \rangle$, and $\langle 110 \searrow \rangle$. Four more states correspond to the input gates of the circuit. Each of these states is a Boolean value, attributed with a

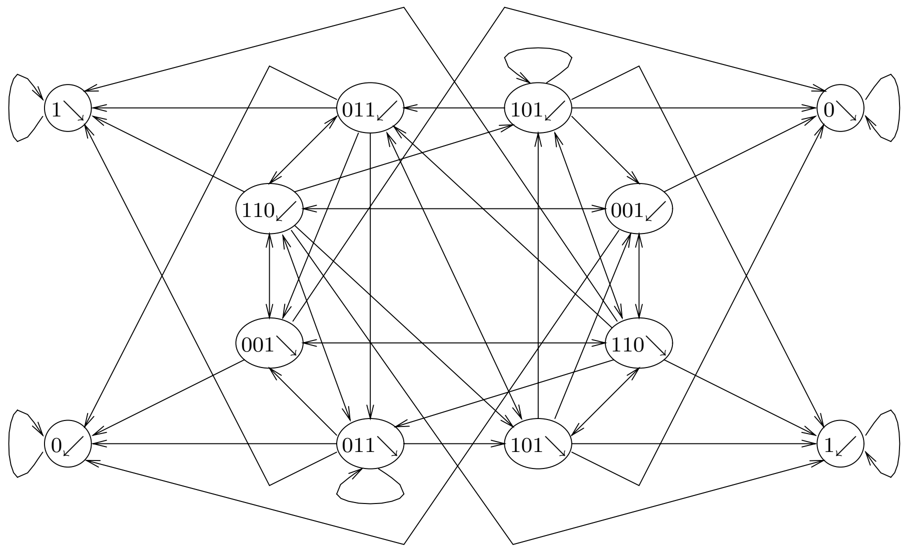

Figure 2: A fixed transition system that embodies all NAND circuits

direction; thus the "input states" are $\langle 0 \swarrow \rangle$, $\langle 1 \swarrow \rangle$, $\langle 0 \searrow \rangle$, and $\langle 1 \searrow \rangle$. The intuition is that an internal state $\langle l, r, val, d \rangle$ corresponds to a NAND gate that has the value l in its left input, has the value r in its right input, and whose output val can be only a d-input of other gates. Similarly, an input state $\langle val, d \rangle$ corresponds to an input gate with output val that can only be a d input of other gates.

Proof of Theorem 4-5 Accordingly, the transitions from an internal state $\langle l, r, val, d \rangle$ correspond to the possible ways of having l and r as right and left inputs, respectively. Thus, we have transitions from this state to all (internal or input) states with either $val = l$ and $d = \swarrow$ or $val = r$ and $d = \searrow$. For example, the internal state $\langle 101 \swarrow \rangle$ has transitions to the states $\langle 001 \swarrow \rangle, \langle 011 \swarrow \rangle, \langle 101 \swarrow \rangle, \langle 110 \searrow \rangle, \langle 1 \swarrow \rangle$, and $\langle 0 \searrow \rangle$. It has transitions from all states $\langle l, r, val, d \rangle$ with $l = 0$. In addition, the input states have self loops.

Proof of Theorem 4-6 We label an internal state by either $\swarrow$ or $\searrow$, according to its directional element. For example, the node $\langle 101 \swarrow \rangle$ is labeled $\{\swarrow\}$. We label an input state by both its value and direction. For example, the node $\langle 1 \searrow \rangle$ is labeled $\{1, \searrow\}$. We define the initial states of S' to be those with $val = 1$, and we impose no fairness condition. Clearly, the size of S' is fixed.

Proof of Theorem 4-7 Given a circuit $\beta = \langle G, G_{in}, g_{out}, T_l, T_r \rangle$ and an input vector $\vec{x}$, the transition system S is simply β, with attributions of directions, and labeling of input gates according to $\vec{x}$. More precisely, we duplicate all gates and inputs of β so that the output of each gate is either always a left input of other gates, in which case we label it with $\swarrow$, or always a right input of other gates, in which case we label it with $\searrow$. In addition, we add self loops to the input gates and label them with their values. As the set of initial states, we take both attributions of g_{out}. Formally, $S = \langle \{\swarrow, \searrow\} \cup \{0,1\} \times \{\swarrow, \searrow\}, (G \cup G_{in}) \times \{\swarrow, \searrow\}, T, \{g_{out}\} \times \{\swarrow, \searrow\}, L \rangle$, where:

- $T(\langle g, d\rangle, \langle g', d'\rangle)$ if and only if one of the following holds:
  1. $g \in G$, $d' = l$, and $T_l(g) = g'$,
  2. $g \in G$, $d' = r$, and $T_r(g) = g'$, or
  3. $g \in G_{in}$, $g = g'$, and $d = d'$.
- For $g \in G$, we have $L(\langle g, d\rangle) = d$. For $g_i \in G_{in}$, we have $L(\langle g_i, d\rangle) = \langle x_i, d\rangle$.

Proof of Theorem 4-8 We define the *depth* of a gate as the length of the longest path from it to an input gate. Note that the only cycles in S are the self loops in the input gates, and the path cannot use them. Thus, for $g \in G_{in}$, we have $depth(g) = 0$, and for $g \in G$, we have $depth(g) = 1 + \max\{depth(T_l(g)), depth(T_r(g))\}$.

Proof of Theorem 4-9 We first prove that for a simulation relation H from S to S' and for every pair $\langle\langle g, d\rangle, \langle val, d'\rangle\rangle$ or $\langle\langle g, d\rangle, \langle l, r, val, d'\rangle\rangle$ in H, the output of the gate g on the vector $\vec{x}$ is val. The proof proceeds by induction on $depth(g)$. If $depth(g) = 0$, then $g \in G_{in}$. Let $g = g_i$. By the definition of L', the state $\langle g_i, d\rangle$ is labeled with x_i and can therefore be related by H only to states $\langle val, d'\rangle$ for which $val = x_i$. Assume that the claim holds for all gates of depth at most i. Let g be such that $depth(g) = i + 1$. Then, $g \in G$, and $\langle g, d\rangle$ is mapped to some internal state $t = \langle l, r, v, d'\rangle$. By the definition of simulation, for every successor $\langle g', d'\rangle$ of $\langle g, d\rangle$ there exists a successor t' of t such that $H(g', t')$. We know that $\langle g, d\rangle$ has two successors, $\langle T_l(g), \swarrow\rangle$ and $\langle T_r(g), \searrow\rangle$. By the definition of S', all the successors of t that are labeled with $\swarrow$ have value l. Therefore, by the induction hypothesis, the output of $T_l(g)$ is l. Similarly, the output of $T_r(g)$ is r. Thus, the output of g is $\text{NAND}(l, r)$. By the definition of S', we also have $v = \text{NAND}(l, r)$, and we are done.

Proof of Theorem 4-10 We now prove that the output of β on $\vec{x}$ is 1 if and only if S simulates S'. Assume first that S simulates S'. Let H be the simulation relation from S to S'. When we take as the initial set of S' the states with $val = 1$, both states $\langle g_{out}, \swarrow \rangle$ and $\langle g_{out}, \searrow \rangle$ of S are related by H to states with $val = 1$. Hence, by the above claim, the output of β on $\vec{x}$ is 1.

Proof of Theorem 4-10 Assume that the output of β on $\vec{x}$ is 1. Consider a relation H from the states of S to the states of S' in which $H(\langle g, d \rangle, \langle l, r, val, d' \rangle)$ for an internal gate g if and only if l is the value of the left input to g, r is the value of the right input to g, val is the output of g on $\vec{x}$, and $d = d'$; and $H(\langle g, d \rangle, \langle val, d' \rangle)$ for an input gate g if and only if val is the output of g on $\vec{x}$ and $d = d'$. We show that H is a simulation relation from S to S'. Consider a state w_1 in S with $H(w_1, w'_1)$. We have to show that for every successor w_2 of w_1 there exists a successor w'_2 of w'_1 such that $H(w_2, w'_2)$. Consider first the case where $w_1 = \langle g, d \rangle$ for $g \in G_{in}$. By the definition of S, the state w_1 has a single successor w_2 with $w_2 = w_1$. Let w'_1 be such that $H(w_1, w'_1)$. By the definition of H, we have $w'_1 = \langle val, d \rangle$, where val is the output of g on $\vec{x}$. By the definition of S', the state w'_1 has a single successor w'_2 with $w'_2 = w'_1$; hence $H(w_2, w'_2)$. Consider now the case where $w_1 = \langle g, d \rangle$ for $g \in G$. Let w'_1 be such that $H(w_1, w'_1)$. By the definition of H, we have $w'_1 = \langle l_1, r_1, val_1, d \rangle$, where l_1 is the value of the left input to g, r_1 is the value of the right input to g, and val_1 is the output of g on $\vec{x}$. By the definition of S, the state w_1 has as successors the two states $\langle T_l(g), \swarrow \rangle$ and $\langle T_r(g), \searrow \rangle$. Consider the state $w_2 = \langle T_l(g), \swarrow \rangle$. Let $w'_2 = \langle l_2, r_2, val_2, \swarrow \rangle$ be a successor of w'_1 for which l_2 is the value of the left input to $T_l(g)$, r_2 is the value of the right input to $T_l(g)$, and val is the output of $T_l(g)$ on $\vec{x}$. By the definition of S', such a successor w'_2 exists. Also, by the definition of H, we have $H(w_2, w'_2)$. The proof for the state $\langle T_r(g), \searrow \rangle$ is similar.

Proof of Theorem 4 □

4 Verification of Fair Transition Systems

4-1 In this section, we study the complexity of the fair-containment and the fair-simulation problems. We show that both problems are PSPACE-complete, and that the implementation complexity of both problems is significantly lower.

4.1 The Fair-Containment Problem

4.1-1 We have seen that the containment problem is PSPACE-complete, and that its implementation complexity is NLOGSPACE-complete. In this section, we check what happens to the complexity when we augment the transition systems with fairness conditions. We start with some auxiliary results.

Lemma 1 $\mathcal{W}(n,m) \rightarrow \mathcal{U}(nm)$.

Proof of Lemma 1 Consider a weakly fair transition system $S = \langle \Sigma, W, R, W_0, L, \alpha \rangle$, with fairness condition $\alpha = \{\langle B_1, G_1 \rangle, \ldots, \langle B_m, G_m \rangle\}$. For every $W' \subseteq W$ and pair $\langle B_i, G_i \rangle$, we have that $W' \cap (W \setminus B_i) = \emptyset$ implies that $W' \cap G_i \neq \emptyset$, if and only if $W' \cap ((W \setminus B_i) \cup G_i) \neq \emptyset$. Hence, a computation π is fair in S if and only if it is fair in all the unconditionally fair transition systems $S_i = \langle \Sigma, W, R, L, W_0, (W \setminus B_i) \cup G_i \rangle$. The result then follows from the known bound on the size of the product of unconditionally fair transition systems [Cho74].

Proof of Lemma 1 □

Lemma 2 *Given a transition system $S \in \mathcal{U}(n)$ with a state space W and a set $B \subseteq W$, we can construct a transition system $S' \in \mathcal{U}(2n)$ such that a trace ρ is accepted by S' if and only if there exists a fair computation π in S such that $\mathrm{Inf}(\pi) \cap B = \emptyset$ and $L(\pi) = \rho$.*

Proof of Lemma 2-1 **Proof of Lemma 2** The idea, as suggested in [Kur87], is that the transition system S' guesses a position in each of its computations from which no state of B can be visited. For that, it maintains two copies of S. The first copy allows visits in states in B. The second copy does not allow visits in states in B. Each computation starts in the first copy, and should eventually move to the second copy. Formally, for $S = \langle \Sigma, W, R, L, W_0, \alpha \rangle$, we define $S' = \langle \Sigma, W \times \{1,2\}, R', L', W_0 \times \{1\}, \alpha \times \{2\} \rangle$, where:

- For every w and w' in W and i and i' in $\{1,2\}$, we have that $R'(\langle w, i \rangle, \langle w', i' \rangle)$ if and only if $R(w, w')$ and either:
  - $w' \notin B$ and $i \leq i'$, or
  - $w' \in B$ and $i = i' = 1$.
- For every $w \in W$ and $i \in \{1,2\}$, we have $L'(\langle w, i \rangle) = L(w)$.

Proof of Lemma 2-2 We prove the correctness of our construction. Consider a trace $\rho \in \Sigma^\omega$, and assume there exists a fair computation $\pi = w_0, w_1, \ldots$ in S such that $Inf(\pi) \cap B = \emptyset$ and $L(\pi) = \rho$. Let w_i be such that for all $j > i$, we have $w_j \notin B$. The computation $\pi' = \langle w_0, 1\rangle, \langle w_1, 1\rangle, \ldots, \langle w_i, 1\rangle, \langle w_{i+1}, 2\rangle, \langle w_{i+2}, 2\rangle, \ldots$ is then fair in S', and ρ is accepted by S'. Assume now that ρ is accepted by S'; thus, there exists a computation $\pi' = \langle w_0, i_0\rangle, \langle w_1, i_1\rangle, \ldots$ in S' such that $L(\pi') = \rho$ and $Inf(\pi') \cap (\alpha \times \{2\}) \neq \emptyset$. As no transitions from states in $W \times \{2\}$ to states in $W \times \{1\}$ are possible, the computation π' eventually gets trapped in states in $W \times \{2\}$. Therefore, as no transitions to states in $B \times \{2\}$ are possible, the computation π' visits states in B only finitely often. Finally, as $R'(\langle w, i\rangle, \langle w', i'\rangle)$ only if $R(w, w')$, the computation $\pi = w_0, w_1, \ldots$ exists and is fair in S, and we are done. Note that S' is not necessarily total. For that, we restrict S' to states that have at least one R'-successor. Clearly, this does not affect the traces of S'.

Proof of Lemma 2 □

Lemma 3 $\mathcal{S}(n, m) \rightarrow \mathcal{U}(n2^{O(m)})$.

Proof of Lemma 3-1 **Proof of Lemma 3** Consider a strongly fair transition system $S = \langle \Sigma, W, R, L, W_0, \alpha\rangle$ with fairness condition $\alpha = \{\langle B_1, G_1\rangle, \ldots, \langle B_m, G_m\rangle\}$. With every $I \subseteq \{1, \ldots, m\}$, we associate an unconditionally fair transition system S_I that accepts the traces $L(\pi)$ of S for which $Inf(\pi) \cap B_i \neq \emptyset$ and $Inf(\pi) \cap G_i \neq \emptyset$ for all $i \in I$, and $Inf(\pi) \cap B_i = \emptyset$ for all $i \notin I$. For that, we first define a weakly fair transition system $S'_I = \langle \Sigma, W, R, W_0, L, \alpha_I\rangle$ that accepts the traces $L(\pi)$ of S for which $Inf(\pi) \cap B_i \neq \emptyset$ and $Inf(\pi) \cap G_i \neq \emptyset$ for all $i \in I$. This is done by defining $\alpha_I = \bigcup_{i \in I}\{\langle W, B_i\rangle, \langle W, G_i\rangle\}$. If $S \in \mathcal{S}(n, m)$, then $S'_I \in \mathcal{W}(n, 2m)$ and hence, by Lemma 1, we can translate it to $S''_I \in \mathcal{U}(2nm)$. Let $B = \bigcup_{i \notin I} B_i$. Clearly, for every computation π of S''_I, we have that $Inf(\pi) \cap B_i = \emptyset$ for all $i \notin I$ if and only if $Inf(\pi) \cap B = \emptyset$. Hence, according to Lemma 2, we can construct the transition system S_I with $4nm$ states.

Proof of Lemma 3-2 We prove that $\mathcal{T}(S) = \bigcup_{I \subseteq \{1,2,\ldots,m\}} \mathcal{T}(S_I)$. A computation π is fair in S if for every pair $\langle B_i, G_i\rangle$, either $Inf(\pi) \cap B_i = \emptyset$, or both $Inf(\pi) \cap B_i \neq \emptyset$ and $Inf(\pi) \cap G_i \neq \emptyset$. Let $f(\pi) \subseteq \{1, \ldots, m\}$ be such that $Inf(\pi) \cap B_i \neq \emptyset$ if and only if $i \in f(\pi)$. It is easy to see that π is fair in S if and only if π is fair in $S_{f(\pi)}$. The lemma now follows from the fact that union is easy for transition systems (e.g., by defining the initial set as union of the initial sets of the underlying systems).

Proof of Lemma 3 □

Theorem 5 *The fair-containment problem $\mathcal{T}(S) \subseteq \mathcal{T}(S')$ for $S \in \bigcup\{\mathcal{U}, \mathcal{W}, \mathcal{S}\}$ and $S' \in \bigcup\{\mathcal{U}, \mathcal{W}, \mathcal{S}\}$ is PSPACE-complete.*

Proof of Theorem 5-1 **Proof of Theorem 5** As there are three possible types for the transition system S and three possible types for the transition system S', we have nine containment problems to solve to prove a PSPACE upper bound. We solve them all using the same method:

1. Translate the transition system S to an unconditionally fair transition system S_U.
2. Construct an unconditionally fair transition system $\overline{S'_U}$ that complements the transition system S'.
3. Check $\mathcal{T}(S_U) \cap \mathcal{T}(\overline{S'_U})$ for emptiness.

Proof of Theorem 5-2 This is how we perform step 1 for the three possible types of S:

1. $\mathcal{U}(n) \rightarrow \mathcal{U}(n)$.
2. $\mathcal{W}(n,m) \rightarrow \mathcal{U}(nm)$ [Lemma 1].
3. $\mathcal{S}(n,m) \rightarrow \mathcal{U}(n2^{O(m)})$ [Lemma 3].

Proof of Theorem 5-3 This is how we perform step 2 for the three possible types of S':

1. $\overline{\mathcal{U}(n)} \rightarrow \mathcal{U}(2^{O(n \log n)})$ [Saf88].
2. $\overline{\mathcal{W}(n,m)} \rightarrow \overline{\mathcal{U}(nm)} \rightarrow \mathcal{U}(2^{O(nm \log(nm))})$ [Saf88].
3. $\overline{\mathcal{S}(n,m)} \rightarrow \mathcal{S}(2^{O(nm \log(nm))}, nm) \rightarrow \mathcal{U}(2^{O(nm \log(nm))})$ [Saf92].

Proof of Theorem 5-4 For all three types of S, going to S_U involves an at-most exponential blowup. Similarly, for all three types of S', going to $\overline{S'_U}$ involves an at-most exponential blowup. Thus, the size of the product of S_U and $\overline{S'_U}$ is exponential in the sizes of S and S' [Cho74]. By [VW94], the nonemptiness problem for unconditionally fair transition systems is in NLOGSPACE. Hence, as

NLOGSPACE = co-NLOGSPACE, checking the product of S_U and $\overline{S'_U}$ for emptiness can be done in space polynomial in their sizes.

Proof of Theorem 5-5 Hardness in PSPACE follows from the PSPACE lower bound for trace containment (Theorem 1). Indeed, we can easily define a fairness condition for which all the computations are fair ($\alpha = W$ for unconditional fairness and $\alpha = \emptyset$ for weak and strong fairness).

Proof of Theorem 5 □

4.1.1 Implementation Complexity

4.1.1-1 Recall that our main concern is the complexity in terms of the (much larger) implementation. We now turn to consider the implementation complexity of fair containment.

Theorem 6 *The implementation complexity of checking $\mathcal{T}(S) \subseteq \mathcal{T}(S')$ for $S \in \bigcup\{\mathcal{U}, \mathcal{W}\}$ and $S' \in \bigcup\{\mathcal{U}, \mathcal{W}, \mathcal{S}\}$ is NLOGSPACE-complete.*

Proof of Theorem 6 In the case where $S \in \bigcup\{\mathcal{U}, \mathcal{W}\}$, the translation of S to S_U involves only a polynomial blowup. Thus, in this case, fixing the size of S', the nondeterministic algorithm described in the proof of Theorem 5, requires space logarithmic in the size of S. Hardness in NLOGSPACE follows from the NLOGSPACE lower bound for the implementation complexity of containment (Theorem 2).

Proof of Theorem 6 □

4.1.1-2 So, for the case where the implementation does not use the strong fairness condition, our fair-containment algorithm requires space that is only polylogarithmic in the size of the implementation. Clearly, this is not the case when the implementation does use the strong fairness condition. Then, our algorithm requires space that is polynomial in the size of the implementation, and time that is exponential in the size of the implementation. We suggest an alternative algorithm that requires time that is only polynomial in the size of the implementation. The price is larger complexity in terms of the size of the specification. We first need the following lemma.

Lemma 4 *For $S_1 \in \mathcal{S}(n_1, m)$ and $S_2 \in \mathcal{U}(n_2)$, there exists $S \in \mathcal{S}(n_1 n_2, m+1)$ such that $\mathcal{T}(S) = \mathcal{T}(S_1) \cap \mathcal{T}(S_2)$.*

Proof of Lemma 4 Given a strongly fair transition system $S_1 = \langle \Sigma, W_1, R_1, W_1^0, L_1, \alpha \rangle$ and an unconditionally fair transition system $S_2 = \langle \Sigma, W_2, R_2, W_2^0, L_2, \beta \rangle$, consider the strongly fair transition system $S = \langle \Sigma, W, R, W_0, L, \gamma \rangle$, where:

- $W = \{\langle w_1, w_2 \rangle \colon w_1 \in W_1, w_2 \in W_2, \text{ and } L_1(w_1) = L_2(w_2)\}$,
- $R(\langle w_1, w_2 \rangle, \langle w_1', w_2' \rangle)$ if and only if $R_1(w_1, w_1')$ and $R_2(w_2, w_2')$,
- $W_0 = (W_1^0 \times W_2^0) \cap W$,
- for every $\langle w_1, w_2 \rangle \in W$, we have $L(\langle w_1, w_2 \rangle) = L_1(w_1)$, and
- $\gamma \subseteq 2^W \times 2^W$ is such that $\langle G, B \rangle \in \gamma$ if and only if either there exists $\langle G', B' \rangle \in \alpha$ for which $G = (G' \times W_2) \cap W$ and $B = (B' \times W_2) \cap W$, or $G = W$ and $B = (W_1 \times \beta) \cap W$.

It is easy to see that S accepts an input trace if and only if both S_1 and S_2 accept it, and that its size is as required.

Proof of Lemma 4 □

Theorem 7 *The implementation complexity of checking $\mathcal{T}(S) \subseteq \mathcal{T}(S')$ for $S \in \mathcal{S}$ and $S' \in \bigcup\{\mathcal{U}, \mathcal{W}, \mathcal{S}\}$ is in PTIME.*

Proof of Theorem 7 Given S and S', we construct, as in the proof of Theorem 5, the unconditionally fair transition system $\overline{S'_U}$. Unlike the algorithm there, we do not translate the transition system S to an unconditionally fair system. Rather, we check the nonemptiness of $\mathcal{T}(S) \cap \mathcal{T}(\overline{S'_U})$. The nonemptiness problem for strongly fair transition systems can be solved in polynomial time [EL85]. Hence, by Lemma 4, we can check the nonemptiness of the intersection in time polynomial in the size of S.

Proof of Theorem 7 □

4.1.1-3 Note that the algorithm presented in the proof of Theorem 7 uses time and space exponential in the size of the specification, in contrast to the algorithm in the proof of Theorem 5 that uses space polynomial in the size of the specification. Nevertheless, as S' is usually much smaller than S, the algorithm in the proof of Theorem 7 may work better in practice. Can we do better and get the NLOGSPACE complexity we have for implementations that use the unconditional or weak fairness conditions? As we now show, the answer to this question is most likely negative. To see this, we first need the following theorem.

Theorem 8 *The nonemptiness problem for strongly fair transition systems is PTIME-hard.*

Proof of Theorem 8-1 **Proof of Theorem 8** We do a reduction from propositional anti-Horn satisfiability (PAHS). Propositional anti-Horn clauses are obtained from propositional Horn clauses by replacing each proposition p with $\neg p$. Thus, a propositional anti-Horn clause is either of the form $p \rightarrow q_1 \vee \ldots \vee q_n$ (an empty disjunction is equivalent to false) or of the form $q_1 \vee \ldots \vee q_n$. As propositional Horn satisfiability is PTIME-complete [Pla84], then clearly, so is PAHS.

Proof of Theorem 8-2 Given an instance I of PAHS, we construct the transition system:

$$S_I = \langle \{a\}, W, W \times W, W, L, \alpha \rangle$$

where:

- $W = Q \times \{s\}$, where Q is the set of all the propositions in I, and $s \notin Q$,
- L maps all states to a, and
- α is the strongly fair condition defined as follows:
  - for a clause $p \rightarrow q_1 \vee \ldots \vee q_n$ in I, we have $\langle \{p\}, \{q_1, \ldots, q_n\} \rangle$ in α, and
  - for a clause $q_1 \vee \ldots \vee q_n$ in I, we have $\langle W, \{q_1, \ldots, q_n\} \rangle$ in α.

Proof of Theorem 8-3 Each computation of S_I induces an assignment to the propositions in I. A proposition is assigned *true* if and only if the computation visits it infinitely often. In addition, for each assignment to the propositions in I, there exists a computation of S_I that induces it (the state s guarantees that the above also holds for the assignment in which all propositions are assigned *false*). The definition of α thus guarantees that I is satisfiable if and only if S_I is nonempty.

Proof of Theorem 8 □

4.1.1-4 We note that the nonemptiness problem for strongly fair transition systems is already PTIME-hard for systems with pairs of a constant size. In 3-PAHS, all the clauses are of the form $p \rightarrow q_1 \vee q_2$, except possibly one clause, which has the form p. It is easy to see that by introducing polynomially many new propositions, every instance I of PAHS can be reduced to an instance I' of 3-PAHS. Given such I, we can construct a strongly fair transition system in which every pair has at most three states in its two sets. The construction is the same as the one suggested in the proof, only that we handle the clause p by $|W|-1$ pairs of the form $\langle\{q\},\{p\}\rangle$, one for each $q \in W \setminus \{p\}$.

4.1.1-5 Therefore, unlike unconditionally or weakly fair transition systems, for which the nonemptiness problem is NLOGSPACE-complete, testing strongly fair transition systems for nonemptiness is PTIME-complete. Theorems 7 and 8 imply the following theorem.

Theorem 9 *The implementation complexity of checking $\mathcal{T}(S) \subseteq \mathcal{T}(S')$ for $S \in \mathcal{S}$ and $S' \in \bigcup\{\mathcal{U}, \mathcal{W}, \mathcal{S}\}$ is PTIME-complete.*

4.2 The Fair-Simulation Problem

4.2.1 The Upper Bound

Theorem 10 *The fair-simulation problem $S \leq S'$ for $S \in \bigcup\{\mathcal{U}, \mathcal{W}, \mathcal{S}\}$ and $S' \in \bigcup\{\mathcal{U}, \mathcal{W}, \mathcal{S}\}$ is in PSPACE.*

Proof of Theorem 10-1 **Proof of Theorem 10** Given $S = \langle\Sigma, W, R, W_0, L, \alpha\rangle$ and $S' = \langle\Sigma, W', R', W_0', L', \alpha'\rangle$, we show how to check in polynomial space that a candidate relation H is a simulation from S to S'. The claim then follows, since we can enumerate all candidate relations using polynomial space. First, we check (easily) that for every $w \in W_0$ there exists $w' \in W_0'$ such that $H(w, w')$. We then check (also easily) that for all $\langle w, w'\rangle \in H$, we have $L(w) = L(w')$. It is left to check that for all $\langle w, w'\rangle \in H$, the pair $\langle w, w'\rangle$ is good in H. To do this, we define, for every $\langle w, w'\rangle \in H$, two transition systems. The alphabet of both systems is W. The first transition system, A_w, accepts all the fair w-computations in S. The second transition system, $U_{w'}$, accepts all the sequences π in W^ω for which there exists a fair w'-computation π' in S' such that $H(\pi, \pi')$. Clearly, the pair $\langle w, w'\rangle$ is good in H if and only if $\mathcal{T}(A_w) \subseteq \mathcal{T}(U_{w'})$.

Proof of Theorem 10-2 We define A_w and $U_{w'}$ as follows. The transition system A_w does nothing but trace the w-computations of S, accepting those that satisfy S's acceptance condition. Formally, $A_w = \langle W, W, R, \{w\}, L'', \alpha \rangle$, where for all $u \in W$, we have $L''(u) = u$.

Proof of Theorem 10-3 The transition system $U_{w'}$ has members of H as its set of states. Thus, each state has two elements. The second element of each state in $U_{w'}$ is a state in W', and according to R', it induces the transitions. The first element in each state of $U_{w'}$ is a state in W, and it induces the labeling. This combination guarantees that a computation $\pi'' \in H^\omega$, whose W' elements form the computation $\pi' \in (W')^\omega$ and whose states are labeled with $\pi \in W^\omega$, satisfies $H(\pi, \pi')$. Formally, $U_{w'} = \langle W, H, R'', W_0'', L''', \alpha'' \rangle$, where:

- R'' is adjusted to the new state space; i.e., $R''(\langle t, t' \rangle, \langle q, q' \rangle)$ if and only if $R'(t', q')$,
- $W_0'' = (W \times \{w'\}) \cap H$,
- for every $\langle t, t' \rangle \in H$, we have $L'''(\langle t, t' \rangle) = t$, and
- α'' is also adjusted to the new state space; i.e., each set $B \subseteq W'$ in α' is replaced by the set $(W \times B) \cap H$ in α''.

Note that R'' is not necessarily total. For that, we restrict $U_{w'}$ to states that have at least one R'' successor. Clearly, this does not affect the traces of $U_{w'}$.

Proof of Theorem 10-4 According to Theorem 5, checking that $\mathcal{T}(A_w) \subseteq \mathcal{T}(U_{w'})$ can be done in space polynomial in the sizes of A_w and $U_{w'}$, thus polynomial in the sizes of S and S'.

Proof of Theorem 10 □

Proof of Theorem 10-5 We note that our algorithm can be easily adjusted to check S and S' for fair bisimulation.

4.2.2 The Lower Bound

4.2.2-1 For a transition system $S = \langle \Sigma, W, R, W_0, L, \alpha \rangle$, we say that S is *universal* if and only if $\mathcal{T}(S) = \Sigma^\omega$. The *universality problem* is to determine whether a given transition system is universal. As we have already mentioned in the proof of Theorem 1, Mayer and Stockmayer prove a PSPACE lower bound for the problem of determining whether a finite-acceptance transition system

S is universal [MS72]. Our PSPACE lower bound for the fair-simulation problem follows the lines of their proofs, and we first give its details, which are easily adjusted to infinite traces.

Theorem 11 *The universality problem, $L(S) = \Sigma^\omega$ for $S \in \bigcup\{\mathcal{U}, \mathcal{W}, \mathcal{S}\}$, is PSPACE-hard.*

Proof of Theorem 11-1

Proof of Theorem 11 We do a reduction from polynomial-space Turing machines. Given a Turing machine T of space complexity $s(n)$, we construct a transition system S_T of size linear in T, and $s(n)$ such that S_T is universal if and only if T does not accept the empty tape. We assume, without loss of generality, that once T reaches a final state, it loops there forever. The system S_T accepts a trace w if and only if w is not an encoding of a legal computation of T over the empty tape, or if w is an encoding of a legal yet rejecting computation of T over the empty tape. Thus, S_T rejects a trace w if and only if it encodes a legal and accepting computation of T over the empty tape. Hence S_T is universal if and only if T does not accept the empty tape.

Proof of Theorem 11-2

We now give the details of the construction of S_T. Let $T = \langle \Gamma, Q, \rightarrow, q_0, F\rangle$, where Γ is the alphabet, Q is the set of states, $\rightarrow \subseteq Q \times \Gamma \times Q \times \Gamma \times \{L, R\}$ is the transition relation (we use $(q, a) \rightarrow (q', b, \Delta)$ to indicate that when T is in state q and it reads the input a in the current tape cell, it moves to state q', writes b in the current tape cell, and its reading head moves one cell to the left/right, according to Δ), q_0 is the initial state, and $F \subseteq Q$ is the set of accepting states. We encode a configuration of T by a string $\#\gamma_1\gamma_2 \ldots (q, \gamma_i) \ldots \gamma_{s(n)}$. That is, a configuration starts with $\#$, and all its other letters are in Γ, except for one letter in $Q \times \Gamma$. The meaning of such a configuration is that the j^{th} cell in T, for $1 \leq j \leq s(n)$, is labeled γ_j, the reading head points at cell i, and T is in state q. For example, the initial configuration of T is $\#(q_0, b)b \ldots b$ (with $s(n) - 1$ occurrences of bs) where b stands for an empty cell. We can then encode a computation of T by a sequence of configurations.

Proof of Theorem 11-3

Let $\Sigma = \{\#\} \cup \Gamma \cup (Q \times \Gamma)$ and let $\#\sigma_1 \ldots \sigma_{s(n)}\#\sigma'_1 \ldots \sigma'_{s(n)}$ be two successive configurations of T. We also set σ_0, σ'_0, and $\sigma_{s(n)+1}$ to $\#$. For each triple $\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1}\rangle$ with $1 \leq i \leq s(n)$, we know by the transition relation of T what σ'_i should be. In addition, the letter $\#$ should repeat exactly every $s(n)+1$ letters. Let $\mathit{next}(\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1}\rangle)$ denote our expectation for σ'_i. That is:

- $Next(\langle \gamma_{i-1}, \gamma_i, \gamma_{i+1} \rangle) = next(\langle \#, \gamma_i, \gamma_{i+1} \rangle) = next(\langle \gamma_{i-1}, \gamma_i, \# \rangle) = \gamma_i$.
- $Next(\langle (q, \gamma_{i-1}), \gamma_i, \gamma_{i+1} \rangle) = next(\langle (q, \gamma_{i-1}), \gamma_i, \# \rangle) =$

$$\begin{cases} \gamma_i & \text{if } (q, \gamma_{i-1}) \to (q', \gamma'_{i-1}, L) \\ (q', \gamma_i) & \text{if } (q, \gamma_{i-1}) \to (q', \gamma'_{i-1}, R) \end{cases}$$

- $Next(\langle \gamma_{i-1}, (q, \gamma_i), \gamma_{i+1} \rangle) = next(\langle \#, (q, \gamma_i), \gamma_{i+1} \rangle) =$ $next(\langle \gamma_{i-1}, (q, \gamma_i), \# \rangle) = \gamma'_i$ where $(q, \gamma_i) \to (q', \gamma'_i, \Delta)$.[3]
- $Next(\langle \gamma_{i-1}, \gamma_i, (q, \gamma_{i+1}) \rangle) = next(\langle \#, \gamma_i, (q, \gamma_{i+1}) \rangle) =$

$$\begin{cases} \gamma_i & \text{if } (q, \gamma_{i+1}) \to (q', \gamma'_{i+1}, R) \\ (q', \gamma_i) & \text{if } (q, \gamma_{i+1}) \to (q', \gamma'_i, L) \end{cases}$$

- $Next(\langle \sigma_{s(n)}, \#, \sigma'_1 \rangle) = \#$.

Consistency with *next* gives us a necessary condition for a trace to encode a legal computation. In addition, the computation should start with the initial configuration.

Proof of Theorem 11-4 To check consistency with *next*, S_T can use its nondeterminism and guess when there is a violation of *next*. Thus, S_T guesses $\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle \in \Sigma^3$, guesses a position in the trace, checks whether the three letters to be read starting this position are σ_{i-1}, σ_i, and σ_{i+1}, and checks whether $next(\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle)$ is not the letter to come $s(n)+1$ letters later. Once S_T sees such a violation, it goes to an accepting sink. To check that the first configuration is not the initial configuration, S_T simply compares the first $s(n)+1$ letters with $\#(q_0, b)b \ldots b$. Finally, checking whether a legal computation is rejecting is also easy; the final configuration has to be rejecting (one with $q \notin F$).

Proof of Theorem 11 □

4.2.2-2 We would like to do a similar reduction to prove that the fair-simulation problem is PSPACE-hard. For every alphabet Σ, let S_Σ be the transition system $\langle \Sigma, \Sigma, \Sigma \times \Sigma, \Sigma, L_\Sigma, \alpha \rangle$, where $L_\Sigma(\sigma) = \sigma$ and α is such that all the

[3] We assume that the reading head of T does not "fall" from the right or the left boundaries of the tape. Thus, the case where $(i = 1)$ and $(q, \gamma_i) \to (q', \gamma'_i, L)$ and the dual case where $(i = 2^n)$ and $(q, \gamma_i) \to (q', \gamma'_i, R)$ are not possible.

computations of S_Σ are fair. That is, S_Σ is a universal transition system in which each state is associated with a letter $\sigma \in \Sigma$ and $\mathcal{T}(S_\Sigma^\sigma) = \sigma \cdot \Sigma^\omega$. For example, $S_{\{a,b\}}$ is the transition system S in Figure 1. It is easy to see that a transition system S over Σ is universal if and only if $\mathcal{T}(S_\Sigma) \subseteq \mathcal{T}(S)$. It is not true, however, that S is universal if and only if $S_\Sigma \leq S$. For example, the transition system S' in Figure 1 is universal, yet $S_{\{a,b\}} \not\leq S'$. Our reduction overcomes this difficulty by defining S_T in such a way that if S_T is universal, then for each of its states w, we have $\mathcal{T}(S_T^w) = L(w) \cdot \Sigma^\omega$. For such S_T, we do have that S_T is universal if and only if $S_\Sigma \leq S_T$. Indeed, a relation that maps a state σ of S_Σ to all the states of S_T that are labeled with σ is a fair simulation.

Theorem 12 *The fair-simulation problem $S \leq S'$ for $S \in \bigcup\{\mathcal{U}, \mathcal{W}, \mathcal{S}\}$ and $S' \in \bigcup\{\mathcal{U}, \mathcal{W}, \mathcal{S}\}$ is PSPACE-hard.*

Proof of Theorem 12-1

Proof of Theorem 12 We prove hardness for the case where S' is a strongly fair transition system with three pairs. The other cases then follow from the linear translation of S' to any $S' \in \bigcup\{\mathcal{U}, \mathcal{W}, \mathcal{S}\}$. As in the previous proof, we do a reduction from polynomial space Turing machines. Given the Turing machine T, let T' be as follows. Whenever T reaches an accepting configuration, T' "cleans" the tape and starts from the beginning (i.e., empty tape and initial state at the left end of the tape). Thus, T accepts the empty tape if and only if T' has an infinite computation, in which case it visits the initial configuration infinitely often.

Proof of Theorem 12-2

We now define a strongly fair transition system S_T as the union of two strongly fair transition systems, S_T^1 and S_T^2, with the following behaviors. Reading a trace ρ, the fair transition system S_T^1 checks for a violation of the transition relation of T' in ρ (by guessing a violation of *next*). If S_T^1 sees a violation, it goes to an accepting sink. Therefore, the acceptance condition of S_T^1 is a single pair $\langle W_1, G\rangle$, where W_1 is the set of states in S_T^1 and G is a clique of $|\Sigma|$ states, each labeled with a different letter, which S_T^1 enters once it sees a violation of *next*. Reading a trace ρ, the fair transition system S_T^2 checks for the occurrence of the initial configuration in ρ. Because the initial configuration starts with # and has no other # in it, it is easy to check its occurrence. If S_T^2 sees the initial configuration, it goes to a rejecting sink. The acceptance condition of S_T^2 is therefore a single pair $\langle B, \emptyset\rangle$, where B is a clique of $|\Sigma|$ states, each labeled with a different letter, which S_T^2 enters once it sees an occurrence of the initial configuration. Assuming the state spaces of S_T^1 and S_T^2 are disjoint, the fair transition system S_T simply has one copy of S_T^1, and one copy of S_T^2; its initial set is the union of the initial sets of S_T^1 and S_T^2; and its fairness condition has the two pairs $\langle W_1, G\rangle$ and $\langle B, \emptyset\rangle$.

Proof of Theorem 12-3

It follows that the fair transition system S_T accepts a trace ρ if ρ violates *next* or never visits the initial configuration. Thus, S_T does not accept a trace ρ if and only if ρ does not violate *next* and it visits the initial configuration of T. Therefore, S_T is universal if and only if T does not accept the empty tape. Indeed, in both cases there exists no accepting computation of T on the empty tape.

Proof of Theorem 12-4

We want, however, more than a universality test. We want to define S_T in such a way that if it is indeed universal, then for each of its states w, we have $\mathcal{T}(S_T^w) = L(w) \cdot \Sigma^\omega$. Let $S_T = \langle \Sigma, W, R, W_0, L, \alpha\rangle$. We assume that $R \cap (W \times W_0) = \emptyset$. Thus, no computation of S_T visits states from W_0 more than once (this can be easily achieved by duplicating states in W_0 that are visited more than once). We define the transition system S'_T by adding to S_T the transitions from all states to all of the initial states, with the requirement that these transitions can be used only finitely often. Accordingly, $S'_T = \langle \Sigma, W, R \cup (W \times W_0), W_0, L, \alpha \cup \langle W_0, \emptyset\rangle\rangle$. We claim the following:

Claim 1 *S'_T is universal if and only if for each $\sigma \in \Sigma$ we have $w_0 \in W_0$ with $L(w_0) = \sigma$, and for each $w \in W$, we have $\mathcal{T}(S_T^w) = L(w) \cdot \Sigma^\omega$.*

Claim 2 *S_T is universal if and only if S'_T is universal.*

Proof of Theorem 12-5

Claim 1 is immediate, and we prove here Claim 2. Clearly, every computation π of S_T is a computation in S'_T. Since no computation in S_T visits W_0 more than once, adding the pair $\langle W_0, \emptyset\rangle$ to the acceptance condition α, we still have that if π is fair in S_T, then it is also fair in S'_T. Hence $\mathcal{T}(S_T) \subseteq \mathcal{T}(S'_T)$; thus, if S_T is universal, so is S'_T. Assume now that S_T is not universal. Consider a trace ρ not accepted by S_T. Recall that ρ does not violate *next*, and it visits the initial configuration of T. In other words, ρ is of the form yx where y is a prefix not violating *next* and x is an infinite computation of T' (the initial configuration of T is the first configuration in x). An infinite computation of T' visits the initial configuration infinitely often. Therefore, all the suffixes of ρ are of that special form! Hence, if ρ is not accepted by S_T, all its suffixes are also not accepted by S_T. We show that this implies that ρ is not accepted by S'_T too. Assume, by way of contradiction, that ρ is accepted by S'_T. Let $\pi = w_0, w_1, \ldots$ by a fair computation in S'_T, with $L(\pi) = \rho$. By the acceptance condition of S'_T, there exists $i \geq 0$ such that $w_i \in W_0$, and for all $j > i$, we have $W_j \notin W_0$. Hence, for all $j \geq i$, we have $R(w_j, w_{j+1})$. Therefore, the computation $\pi^i = w_i, w_{i+1}, \ldots$ is a fair computation in S_T, and the trace $L(\pi^i)$ is accepted by S_T, contradicting the fact it is a suffix of a trace not accepted by S_T.

Proof of Theorem 12-6 As discussed above, Claims 1 and 2 now imply that $S_\Sigma \leq S'_T$ if and only if S_T is universal; thus $S_\Sigma \leq S'_T$ if and only if T does not accept the empty tape. Because the fairness condition of S_Σ can be specified in terms of either unconditional, weak, or strong fairness, we are done.

Proof of Theorem 12 □

4.2.2-3 Theorems 10 and 12 together imply the following.

Theorem 13 *The fair-simulation problem $S \leq S'$ for $S \in \bigcup\{\mathcal{U}, \mathcal{W}, \mathcal{S}\}$ and $S' \in \bigcup\{\mathcal{U}, \mathcal{W}, \mathcal{S}\}$ is PSPACE-complete.*

4.2.3 Implementation Complexity

4.2.3-1 As our discussions thus far show, fair simulation and fair containment have the same complexity. In Theorem 14, we show that when we consider the implementation complexity of fair simulation, the picture is different. Here, checking implementations that use the unconditional or weak fairness conditions is not easier than checking implementations that use the strong fairness condition; hence fair simulation is most likely harder than fair containment, and the trace-based approach is more efficient.

Theorem 14 *The implementation complexity of checking $S \leq S'$ for $S \in \bigcup\{\mathcal{U}, \mathcal{W}, \mathcal{S}\}$ and $S' \in \bigcup\{\mathcal{U}, \mathcal{W}, \mathcal{S}\}$ is PTIME-complete.*

Proof of Theorem 14-1 **Proof of Theorem 14** We start with the upper bound. Consider the algorithm presented in the proof of Theorem 10. It checks whether a candidate relation H is a simulation. Once we fix S', then, by Theorems 6 and 9, the complexity of checking each pair in the candidate relation is NLOGSPACE for $S \in \bigcup\{\mathcal{U}, \mathcal{W}\}$, and is PTIME for $S \in \mathcal{S}$. Once we fix S', the number of pairs in each candidate relation is linear in the size of S. Thus, fixing S', the problem of checking a candidate relation H is in PTIME. Instead of guessing a relation H and checking it, we do a fixed-point computation as follows [Mil90]. Let

$$H_0 = \{\langle w, w' \rangle : w \in W, w' \in W', \text{ and } L(w) = L(w')\}$$

Thus, H_0 is the maximal relation that satisfies condition 1 of fair simulation. Consider the monotonic function $f : 2^{W \times W'} \rightarrow 2^{W \times W'}$, where

$$f(H) = H \cap \{\langle w, w' \rangle : \langle w, w' \rangle \text{ is good in } H\}$$

Thus, $f(H)$ contains all the pairs in H that are good with respect to the relation H. Let $H^\star$ be the greatest fixed point of f when restricted to pairs in H_0. That is, $H^\star = \nu z.H_0 \cap f(z)$.

Proof of Theorem 14-2 We now prove that $S \leq S'$ if and only if for every $w \in W_0$, we have $(\{w\} \times W'_0) \cap H^\star \neq \emptyset$. First, as $H^\star$ is a fair-simulation relation, the direction from right to left is immediate from the definition of fair simulation. Assume that $S \leq S'$. Then, there exists a fair-simulation relation H' such that for every $w \in W_0$, we have $(\{w\} \times W'_0) \cap H' \neq \emptyset$. Let $H_i = f^i(H_0)$. We show that for every $i \geq 0$ we have $H' \subseteq H_i$. Thus, in particular, $H' \subseteq H^\star$, and we are done. The proof proceeds by induction on i. First, since H' satisfies condition 2 of fair-simulation relations, then clearly $H' \subseteq H_0$. Assume that $H' \subseteq H_i$, and assume by way of contradiction that $H' \not\subseteq H_{i+1}$. Then there exists $\langle w, w' \rangle \in H' \setminus H_{i+1}$. Since $H' \subseteq H_i$, it follows that the pair $\langle w, w' \rangle$ is not good in H_i, which implies, again by the containment of H' in H_i, that it is also not good in H'. Then, however, H' does not satisfy condition 3 of fair simulation, and we reach a contradiction.

Proof of Theorem 14-3 We now consider the complexity of calculating $H^\star$. Since $W \times W'$ is finite, we can calculate H iteratively, starting with H_0 until we reach a fixed point. Because f is monotonic, we must iterate it at most polynomially many times. Hence, out of the $2^{|W \times W'|}$ candidate relations for simulation, we actually check at most $|W \times W'|$ relations. Recall that if S' is fixed, the problem of checking a candidate relation is in PTIME. Also, if S' is fixed, we have only linearly many candidate relations to check. The problem is therefore in PTIME.

Proof of Theorem 14-4 Hardness in PTIME follows from the lower bound in Theorem 4.

Proof of Theorem 14 □

5 Discussion

5-1 We have examined the trace-based and the tree-based approaches to implementation from a complexity-theoretic point of view. Our results show that when we model the specification and the implementation by fair transition systems, the complexity of checking the correctness of a trace-based implementation coincides with that of checking the correctness of a tree-based implementation. Furthermore, when we consider the implementation complexity, then checking implementations that use the unconditional or weak fairness condition is easier in the trace-based approach. Overall, it seems that the trace-based approach is advantageous.

5-2 It is interesting to compare our results with the known complexities of LTL and $\forall$CTL* model checking. Trace-based implementations are part of the linear-time paradigm, and correspond to LTL model checking. Tree-based implementations are part of the branching-time paradigm, and correspond to $\forall$CTL* model checking. All four problems are PSPACE-complete [SC85, EL85]. The model-checking algorithm of $\forall$CTL* uses as a subroutine the model-checking algorithm of LTL [EL85]. In a similar manner, our fair-simulation algorithm uses as a subroutine the fair-containment algorithm. Clearly, the implementation dichotomy and the temporal-logic dichotomy have a lot in common. When we turn to consider the program complexity of model checking, which is the analog to our implementation complexity, this is no longer true. The program complexity of model checking for both LTL and $\forall$CTL* is NLOGSPACE-complete [VW86, BVW94]. In contrast, we see here that implementation is easier in the trace-based approach.

5-3 Our results are summarized in Table 1. All the complexities in the table denote tight bounds.

	Fair Containment	Implementation Complexity of Fair Containment	Fair Simulation	Implementation Complexity of Fair Simulation
S and S' with No Fairness	PSPACE [Theorem 1]	NLOGSPACE [Theorem 2]	PTIME [Theorem 3]	PTIME [Theorem 4]
$S \in \bigcup\{\mathcal{U}, \mathcal{W}\}$ and $S' \in \bigcup\{\mathcal{U}, \mathcal{W}, \mathcal{S}\}$	PSPACE [Theorem 5]	NLOGSPACE [Theorem 6]	PSPACE [Theorem 13]	PTIME [Theorem 14]
$S \in \mathcal{S}$ and $S' \in \bigcup\{\mathcal{U}, \mathcal{W}, \mathcal{S}\}$	PSPACE [Theorem 5]	PTIME [Theorem 9]	PSPACE [Theorem 13]	PTIME [Theorem 14]

Table 1: Is S a correct implementation of S'?

Acknowledgment of support: Part of this work was done in Bell Laboratories during the DIMACS Special Year on Logic and Algorithms.

Orna Kupferman's work was supported in part by the Office of Naval Research Young Investigator Award N00014-95-1-0520, by the National Science Foundation CAREER Award CCR-9501708, by the National Science Foundation grant CCR-9504469, by the Air Force Office of Scientific Research contract F49620-93-1-0056, by the Army Research Office MURI grant DAAH-04-96-1-0341, by the Advanced Research Projects Agency grant NAG2-892, and by the Semiconductor Research Corporation contract 95-DC-324.036.

Moshe Y. Vardi's work was supported in part by the National Science Foundation grants CCR-9628400 and CCR-9700061, and by a grant from the Intel Corporation.

References

[AL91] M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.

[ASB+94] A. Aziz, V. Singhal, F. Balarin, R. Brayton, and A. L. Sangiovanni-Vincentelli. Equivalences for fair Kripke structures. In *Proceedings of the 21st International Colloquium on Automata, Languages and Programming*, Jerusalem, Israel, July 1994.

[AV95] H. R. Andersen and B. Vergauwen. Efficient checking of behavioural relations and modal assertions using fixed-point inversion. In *Computer Aided Verification, Proceedings of the 7th International Conference*, volume 939 of *Lecture Notes in Computer Science*, pages 142–154, Berlin, July 1995. Springer-Verlag.

[BBLS92] S. Bensalem, A. Bouajjani, C. Loiseaux, and J. Sifakis. Property preserving simulations. In *Proceedings of the 4th Conference on Computer Aided Verification*, volume 663 of *Lecture Notes in Computer Science*, pages 260–273, Berlin, June 1992. Springer-Verlag.

[BCG88] M. C. Browne, E. M. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59:115–131, 1988.

[BGS92] J. Balcazar, J. Gabarro, and M. Santha. Deciding bisimilarity is P-complete. *Formal Aspects of Computing*, 4(6):638–648, 1992.

[BVW94] O. Bernholtz, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In D. L. Dill, editor, *Computer Aided Verification, Proceedings of the 6th International Conference*, volume 818 of *Lecture Notes in Computer Science*, pages 142–155, Berlin, June 1994. Springer-Verlag.

[CD88] E. M. Clarke and I. A. Draghicescu. Expressibility results for linear-time and branching-time logics. In *Proceedings of the Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 428–437, Berlin, 1988. Springer-Verlag.

[CDK93] E. M. Clarke, I. A. Draghicescu, and R. P. Kurshan. A unified approach for showing language containment and equivalence between various types of ω-automata. *Information Processing Letters*, 46:301–308, 1993.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.

[Cho74] Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *Journal of Computer and System Sciences*, 8:117–141, 1974.

[EL85] E. A. Emerson and C.-L. Lei. Temporal model checking under generalized fairness constraints. In *Proceedings of the 18th Hawaii International Conference on System Sciences*, North Hollywood, CA, 1985. Western Periodicals Company.

[EL87] E. A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8:275–306, 1987.

[Eme90] E. A. Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science*, pages 997–1072, 1990.

[GHR95] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits of Parallel Computation*. Oxford University Press, 1995.

[GJ79] M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. Freeman and Co., San Francisco, 1979.

[GL94] O. Grumberg and D. E. Long. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems*, 16(3):843–871, 1994.

[Gol77] L. M. Goldschlager. The monotone and planar circuit value problems are log space complete for P. *SIGACT News*, 9(2):25–29, 1977.

[Hen85] M. Hennessy. *Algebraic Theory of Processes*. Cambridge, MA, 1985. MIT Press.

[HHK95] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *Proceedings of the 36th Symposium on Foundations of Computer Science*, pages 453–462, Los Alamitos, CA, 1995. IEEE Computer Society Press.

[Imm88] N. Immerman. Nondeterministic space is closed under complement. *SIAM Journal on Computing*, 17:935–938, 1988.

[Jon75] N. D. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11:68–75, 1975.

[Kel76] R. M. Keller. Formal verification of parallel programs. *Communications of the ACM*, 19:371–384, 1976.

[Kur87] R. P. Kurshan. Complementing deterministic Büchi automata in polynomial time. *Journal of Computer and System Science*, 35:59–71, 1987.

[Kur94] R. P. Kurshan. *Computer Aided Verification of Coordinating Processes*. Princeton, NJ, 1994. Princeton University Press.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the 12th ACM Symposium on Principles of Programming Languages*, pages 97–107, New York, January 1985. ACM.

[LPS81] D. Lehman, A. Pnueli, and J. Stavi. Impartiality, justice, and fairness—the ethics of concurrent termination. In *Proceedings of the 8th Colloquium on Automata, Programming, and Languages (ICALP)*, volume 115 of *Lecture Notes in Computer Science*, pages 264–277, Berlin, July 1981. Springer-Verlag.

[LS84] S. S. Lam and A. U. Shankar. Protocol verification via projection. *IEEE Transactions on Software Engineering*, 10:325–342, 1984.

[LT87] N. A. Lynch and M. R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the 6th ACM Symposium on Principles of Distributed Computing*, pages 137–151, New York, 1987. ACM.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, pages 481–489, London, September 1971. British Computer Society.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Berlin, 1980. Springer-Verlag.

[Mil89] R. Milner. *Communication and Concurrency*. Englewood Cliffs, NJ, 1989. Prentice-Hall.

[Mil90] R. Milner. Operational and algebraic semantics of concurrent processes. *Handbook of Theoretical Computer Science*, pages 1201–1242, 1990.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Berlin, January 1992. Springer-Verlag.

[MS72] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In

Proceedings of the 13th IEEE Symposium on Switching and Automata Theory, pages 125–129, New York, 1972. IEEE Computer Group.

[Pla84] D. A. Plaisted. Complete problems in the first-order predicate calculus. *Journal of Computer and System Sciences*, 29(1):8–35, 1984.

[Pnu85] A. Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In *Proceedings of the 12th International Colloquium on Automata, Languages and Programming*, *Lecture Notes in Computer Science*, pages 15–32, Berlin, 1985. Springer-Verlag.

[Saf88] S. Safra. On the complexity of ω-automata. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*, pages 319–327, Los Alamitos, CA, October 1988. IEEE Computer Society Press.

[Saf92] S. Safra. Exponential determinization for ω-automata with strong-fairness acceptance condition. In *Proceedings of the 24th ACM Symposium on Theory of Computing*, New York, May 1992. ACM.

[SC85] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logic. *Journal of the ACM*, 32:733–749, 1985.

[SVW87] A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[Sze88] R. Szelepcsinyi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.

[Tho90] W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, pages 165–191, 1990.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331, Los Alamitos, CA, June 1986. IEEE Computer Society Press.

[VW94] M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.