

Reasoning about Systems with Transition Fairness

Benjamin Aminof¹, Thomas Ball², and Orna Kupferman¹

¹ Hebrew University, School of Engineering and Computer Science, Jerusalem 91904, Israel
Email: {benj,orna}@cs.huji.ac.il

² Microsoft Research, One Microsoft way, Redmond, WA 98052, USA
Email: tball@microsoft.com

Abstract. Formal verification methods model systems by Kripke structures. In order to model live behaviors of systems, Kripke structures are augmented with *fairness conditions*. Such conditions partition the computations of the systems into fair computations, with respect to which verification proceeds, and unfair computations, which are ignored. Reasoning about Kripke structures augmented with fairness is typically harder than reasoning about non-fair Kripke structures. We consider the *transition fairness* condition, where a computation π is fair iff each transition that is enabled in π infinitely often is also taken in π infinitely often. Transition fairness is a natural and useful fairness condition. We show that reasoning about Kripke structures augmented with transition fairness is not harder than reasoning about non-fair Kripke structures. We demonstrate it for fair CTL and LTL model checking, and the problem of calculating the dominators and postdominators.

1 Introduction

In *formal verification*, we check that a system is correct with respect to a desired behavior by checking that a mathematical model of the system satisfies a formal specification of the behavior. In *model checking*, we model the system by a *Kripke structure*, whose states correspond to configurations of the system, and we specify the desired behavior by means of a *temporal-logic* formula. The model-checking problem is then to decide, given a Kripke structure K and a temporal-logic formula ψ , whether K satisfies ψ [CE81, QS81]. Symbolic methods, abstraction, compositional methods, and many more heuristics have made model checking a successful verification methodology, used in industrial design of both hardware and software [CGP99].

Kripke structures describe only the *safe* behaviors of systems. In order to model *live* behaviors, we have to augment Kripke structures with *fairness conditions*. Such conditions partition the set of infinite paths of a Kripke structure K (and thus also the set of infinite computations of the system that K models) into fair and unfair computations [Fra86, MP92]. For example, when the Kripke structure models a concurrent system, we may wish to restrict attention only to computations in which the scheduler enables each process to proceed infinitely many times.

The model-checking problem can be adjusted to the fair setting: in the *linear-time* approach, specifications describe the desired behavior of all the computations of the system, and thus refer to the language induced by the Kripke structure. Then, fair model

checking amounts to verifying that all the fair computations satisfy the desired behavior. In the *branching-time* approach, specifications may contain the path quantifiers A (for all paths) and E (exists a path) and thus refer to the tree obtained by unwinding the Kripke structure. Then, in fair model checking, path quantification ranges over fair paths only [CES86]. The transition to the fair setting involves a computational price. The exact complexity of the fair model-checking problem depends on the specification formalism and the specific fairness condition that is used.

We consider the fairness condition in which a computation π is fair iff each transition that is enabled in π infinitely often is also taken in π infinitely often. That is, for every transition $\langle w, w' \rangle$ in the Kripke structure, if the path π visits the state w infinitely many times, then π should also have infinitely many visits to w that are immediately followed by a visit to w' . We refer to this type of fairness as *transition fairness*. The transition fairness condition is related to the classical *strong-fairness* condition [MP92] (*Streett fairness*)¹, and is a very natural one in reasoning about concurrent systems [LPS81,KPSZ02].

In order to see why transition fairness is so natural and useful, let us consider the possible sources for nondeterminism in a Kripke structure. We distinguish between *external* and *internal* nondeterminism [Hoa85]. External nondeterminism is caused by the interaction of the system with its environment, and the fact that different inputs are expected in each state of the system. Then, transition fairness requires that in every state of the interaction that is visited infinitely often, the environment provides each input infinitely often. Internal nondeterminism is caused by abstraction. In the case of a concurrent composition between processes, we usually abstract the scheduler. Then, transition fairness restricts attention to a scheduling policy in which for each process, if a process is enabled in a state infinitely often, then it is also scheduled to proceed infinitely often. In case we abstract other factors of the system, like information that is irrelevant for the specification, transition fairness is less natural and restricts attention to computations that are fair with respect to the abstracted information. For an extensive study of transition fairness, further motivation for it, and related definitions of fairness, see [QS83].

Reasoning about systems augmented with the strong-fairness condition is not easy, and is related to the emptiness problem for nondeterministic Streett automata. While the latter problem is PTIME-complete [EL87,KV98a], no linear solutions are known, for both the enumerative and symbolic approaches. The best enumerative algorithm makes use of a lock-step search, used in dynamic graph algorithms, and is at most quadratic in the size of the automaton [HT96]. The best symbolic algorithm is based on the improved algorithm for detecting strongly connected components and requires $O(n(k + \log n))$ symbolic steps, for an automaton of size n and a strong-fairness condition with k pairs

¹ The Streett fairness condition consists of a set of pairs $\langle L_i, U_i \rangle$, with L_i and U_i being a subset of the state space. A path π is fair if for all pairs, if the set L_i is visited infinitely often, then so is U_i . By having a pair $\langle \{w\}, \{w'\} \rangle$ for each transition $\langle w, w' \rangle$ in the Kripke structure, we can encode transition fairness with the Streett fairness condition (a technical issue is the fact w' may be reached via a different transition. This can be easily solved by adding an internal state to each transition). In Section 2.2 we discuss other related fairness conditions.

[KPR98,BGS00]². The complexity of model checking a specification φ with respect to a Kripke structure K augmented with a strong-fairness condition with k pairs is then the complexity of checking emptiness for an automaton of size $|K|2^{|\varphi|}$, in the case of LTL³, and $|\varphi|$ repeated checks for an automaton of size $|K|$, in the case of CTL.

We study the problem of reasoning about systems augmented with the transition fairness condition, and show that this special case of the strong-fairness condition is considerably simpler. We consider the CTL and LTL model-checking problems, as well as the related problem of finding *dominators* and *postdominators* [C⁺91] in directed graphs.

For model checking, we show that fair CTL model checking can be reduced to non-fair CTL model checking. To do so, we introduce a function f with the following properties. Given a CTL formula φ , the function f maps φ to another CTL formula $f(\varphi)$ such that for every Kripke structure K and state w in it, w satisfies φ fairly iff w satisfies $f(\varphi)$ non-fairly. The size of $f(\varphi)$ is linear in the size of φ . The complexity of the algorithm that follows then coincides with the one for non-fair CTL model checking. Since the function f is simple, it is easy to implement it on top of both enumerative and symbolic CTL model-checking tools. We note that another reduction of fair model checking to unfair model checking by a transformation of the specification is described in [QS83]. The logic studied there contains operators corresponding to CTL's EF ("potential reachability") and AF ("inevitable reachability") and the transformation results in formulas that are not CTL formulas.

In order to handle LTL model checking, we introduce a function g that maps LTL formulas θ of some specific forms to a CTL formula $g(\theta)$ such that a fair Kripke structure K has a path that fairly satisfies θ iff K fairly satisfies $g(\theta)$, which in turn can be reduced to checking whether K satisfies $f(g(\theta))$. The forms of θ we handle are those that correspond to the Büchi, Rabin, and Streett acceptance conditions for automata on infinite words. In the automata-theoretic approach to LTL model checking [VW94], we translate an LTL formula ψ into an automaton $\mathcal{A}_{\neg\psi}$ that accepts exactly all the computations that violate ψ . Model checking is then reduced to checking the emptiness of the product of K with $\mathcal{A}_{\neg\psi}$. When K is augmented with a fairness condition, checking the emptiness of the product of K with $\mathcal{A}_{\neg\psi}$ corresponds to checking whether the product contains a path whose projection on the states of K is fair, and which satisfies a formula θ induced by the acceptance condition of $\mathcal{A}_{\neg\psi}$, and is therefore of one of the forms above. Our algorithm can therefore handle many LTL model-checking instances. On the other hand, we show that taking the product of K with $\mathcal{A}_{\neg\psi}$ does not preserve transition fairness, thus our algorithm does not handle all LTL model-checking instances.

The problem of finding the *postdominators* (or, dually, the *dominators*) of a given state in the Kripke structure has application in the area of program analysis and compiler optimizations [C⁺91]. In this application area, the control-flow graph of a program is the Kripke structure, where states correspond to basic blocks in the program and transitions correspond to control-flow transitions between basic blocks in the pro-

² We note that while it is possible to detect SCC also in linearly many symbolic steps [GPP03], it is not known whether the technique there can be applied for an efficient nonemptiness check of Streett automata.

³ See [LH00] for the case the system is modeled by a petri net.

gram. Dominators form the basis of an intermediate representation known as static single assignment form [C⁺91], widely used in optimizing compilers. Computation of postdominators is required for the computation of control dependences in the program dependence graph [FOW87], a data structure used for the automatic parallelization of programs as well as program slicing.

For a state w of a Kripke structure, the set of postdominators of w , denoted $pd(w)$, is the set of states s such that all paths from w eventually reach s . Dually, the set of dominators of w , denoted $dom(w)$, is the set of states s such that all paths to w pass through s . The definition of $pd(w)$ and its applications are of interest mainly in Kripke structures augmented with fairness. Indeed, it makes sense to require s to be reachable only along fair paths, in particular, paths that eventually reach a halting or an error state, or paths that are fair with respect to our transition fairness condition. That is, the definition of $pd(w)$ as used in compiler optimizations generally assumes that loops eventually terminate.

It is easy to show that if x postdominates w and $x \neq w$ then $pd(x) \subset pd(w)$. This means that the postdomination relation can be represented as a tree, where x is the parent of w in the tree iff $pd(w) = \{w\} \cup pd(x)$ (in this case, x is called the “immediate postdominator” of w). There are several efficient algorithms for calculating dominator and postdominator trees [LT79,BKAW98]. Once the tree has been computed, the set $pd(x)$ can be enumerated in time proportional to $|pd(x)|$, for all states x . However, the above techniques maintain an explicit representation of the Kripke structure as a directed graph and explicitly represent the postdominator tree. We use the technique we developed for fair CTL model checking in order to describe a symbolic algorithm that efficiently computes an ROBDD of pairs $\langle w, s \rangle$ such that $s \in pd(w)$. Such a representation is useful because in the application domain of program analysis it often is necessary to enumerate $pd(w)$ for many w without knowing in advance which w will be queried.

Due to lack of space, this version does not contain all the proofs. For the full version, see [ABK04].

2 Preliminaries

2.1 Temporal Logics

We describe the desired behavior of systems by temporal-logic formulas. The logic *LTL* is a linear temporal logic. Formulas of LTL describe computations and are constructed from a set AP of atomic propositions using the usual Boolean operators and the temporal operators X (“next”) and U (“until”). Formally, given a set AP , an LTL formula is one of the following:

- **true**, **false**, or p , for $p \in AP$.
- $\neg\psi_1$, $\psi_1 \vee \psi_2$, $X\psi_1$, or $\psi_1 U \psi_2$, where ψ_1 and ψ_2 are LTL formulas.

We also use the abbreviations \wedge , \rightarrow , and \leftrightarrow , interpreted in the usual way, $F\psi = \mathbf{true} U \psi$ (“eventually”), $G\psi = \neg F\neg\psi$ (“always”), and $\psi_1 W \psi_2 = \psi_1 U \psi_2 \vee G\psi_1$ (“weak until”).

The logic *CTL* is a branching temporal logic. In CTL, we precede each temporal operator by a path quantifier, either E (“for some path”) or A (“for all paths”). Thus, a CTL formula is either:

- **true**, **false**, or p , for $p \in AP$.
- $\neg\varphi_1$ or $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are CTL formulas.
- $EX\varphi_1$ or $AX\varphi_1$, where φ_1 is a CTL formula.
- $E\varphi_1U\varphi_2$ or $A\varphi_1U\varphi_2$, where φ_1 and φ_2 are CTL formulas.

Note that the G and F abbreviations used in LTL can be used also in CTL, i.e., $AF\varphi_1 = A\mathbf{true}U\varphi_1$, and so can the W abbreviation. Thus, while $E\varphi_1W\varphi_2 = E(\varphi_1U\varphi_2 \vee G\varphi_1)$ is not a CTL formula, it is equivalent to the CTL formula $E\varphi_1U\varphi_2 \vee EG\varphi_1$. Similarly, $A\varphi_1W\varphi_2$ is equivalent to the CTL formula $\neg E(\neg\varphi_2)U(\neg\varphi_1 \wedge \neg\varphi_2)$. Accordingly, we refer to EW and AW as legal CTL modalities. The *size* $|\varphi|$ of an LTL or a CTL formula φ is the number of its subformulas. Note that φ can be represented by a DAG with $|\varphi|$ vertices.

We define the semantics of temporal-logic formulas with respect to *Kripke structures*, with which we model systems. A Kripke structure $K = \langle AP, W, R, W_0, L \rangle$ consists of a set AP of atomic propositions, a set W of states, a set $W_0 \subseteq W$ of initial states, a transition relation $R \subseteq W \times W$ that is total in its first element (i.e., for each $w \in W$ there exists at least one w' with $R(w, w')$), and a labeling function $L : W \rightarrow 2^{AP}$, which maps each state to the set of atomic propositions true in this state. When $R(w, w')$, we say that w' is a *successor* of w . A *path* in K represents a *computation* of the system modeled by the Kripke structure and is a (possibly finite) sequence of states $\pi = w_0, w_1, \dots$, such that for all $i \geq 0$, we have $R(w_i, w_{i+1})$. The path π is *initial* if $w_0 \in W_0$. We use π^i to denote the suffix w_i, w_{i+1}, \dots of π , and we use $\pi[i]$ to denote the i 'th state in π . The set of states that π visits is denoted by $visit(\pi)$, and the set of states that π visits infinitely often is denoted by $inf(\pi)$. Formally, $visit(\pi) = \{w : w = w_j \text{ for some } j\}$, and $inf(\pi) = \{w : w = w_j \text{ for infinitely many } j\}$. Note that $inf(\pi) \subseteq visit(\pi)$.

Recall that LTL is a linear temporal logic, thus its formulas are interpreted over paths of the Kripke structure. We use $K, \pi \models \psi$ to indicate that the LTL formula ψ holds along the path π of K . On the other hand, CTL is a branching temporal logic and its formulas are interpreted over states of the Kripke structure. We use $K, w \models \varphi$ to indicate that the CTL formula φ holds in the state w of K . When K is clear from the context, we simply write $\pi \models \psi$ or $w \models \varphi$. For the definition of the relation \models see [Eme90]. We say that a Kripke structure K satisfies an LTL formula ψ , denoted $K \models \psi$, if all the initial paths of K satisfy ψ . Likewise, K satisfies a CTL formula φ , denoted $K \models \varphi$, if all the initial states of K satisfy φ .

A Kripke structure $K = \langle AP, W, R, W_0, L \rangle$ induces a directed graph $G_K = \langle W, R \rangle$. Notations and definitions from graph theory are then applied to Kripke structures in a straightforward way. A state w is *reachable* from a state v iff there is a finite (possibly empty) path w_0, w_1, \dots, w_n such that $v = w_0$ and $w = w_n$ and for every $0 \leq i < n$, we have $R(w_i, w_{i+1})$. A *strongly connected component* (SCC, for short) is a subset C of W such that every state in C is reachable from every other state in C , via states in C . A *maximal strongly connected component* (MSCC, for short) is an SCC C that is

maximal in the sense that we cannot add states to it and still have an SCC. Note that the MSCCs partition W .

Given two MSCC's C and C' we say that $C \leq C'$ if there are states $v \in C'$ and $w \in C$ such that w is reachable from v . The relation \leq defined above constitutes a partial ordering of the set of MSCCs of K . A MSCC C is called a *bottom MSCC* if for every MSCC C' , we have that either C and C' are not comparable or $C \leq C'$. Note that all the bottom MSCCs in a Kripke structure are not trivial (i.e., they contain at least one edge) since R is total in its first element.

It is not hard to see that for every state $v \in W$ there is at least one bottom MSCC C such that for every state $w \in C$, we have that w is reachable from v . Observe that since there are no trivial bottom MSCCs we can assume that every w above is reachable from v using a non-empty path.

2.2 Fairness

A *fairness condition* on a Kripke structure K partitions the paths of K into *fair* and *unfair* paths. The Büchi, Rabin, and Streett acceptance conditions for word automata (for a survey on word automata see [Tho90]) can also be viewed as fairness conditions on Kripke structures. For example, a path π of K is fair with respect to a Büchi condition $\alpha \subseteq W$ iff π visits states in α infinitely often. In this paper we consider *transition fairness*, defined as follows:

- A path π is fair with respect to the transition fairness condition iff all the transitions that are enabled along π infinitely often are also taken along π infinitely often. Formally, a transition $\langle v, w \rangle \in R$ is *enabled* in position i along π , if $\pi[i] = v$. It is *taken*, if in addition $\pi[i + 1] = w$. Thus, equivalently, a path π is fair with respect to the transition fairness condition iff for all $v \in W$, if $v \in \text{inf}(\pi)$ and $R(v, w)$ then $\pi[i] = v$ and $\pi[i + 1] = w$ for infinitely many i 's.

A *fair Kripke structure* $\mathcal{K} = \langle K, \alpha \rangle$ consists of a Kripke structure K and a fairness condition α for K (note that in the case of transition fairness, there is no need to specify a specific α , thus α is some flag indicating the type of fairness). The semantics of LTL and CTL is adjusted to fair Kripke structures by letting path quantification range over fair paths only. For example, an LTL formula ψ is *fairly satisfied* in \mathcal{K} , denoted $\mathcal{K} \models_F \psi$, if all the fair initial paths of K satisfy ψ . For CTL, we use $\mathcal{K}, w \models_F \varphi$ to indicate that a CTL formula φ is fairly satisfied in the state w of the fair Kripke structure \mathcal{K} , and use $\mathcal{K} \models_F \varphi$ to indicate that φ is fairly satisfied in all the initial states of K . For details, see [CGP99].

Note that the transition fairness condition is related to the *successor fairness* condition, where a path π is fair iff all the successors of a state in π that is visited infinitely often are also visited infinitely often. That is, π is fair iff for all $v \in W$, if $v \in \text{inf}(\pi)$ and $R(v, w)$ then $w \in \text{inf}(\pi)$. Successor fairness is a special case of the Streett fairness condition, and, like transition fairness, is used in order to ignore paths that get stuck in a MSCC that is not a bottom MSCC (for an application in the probabilistic setting see [Var85]). Transition fairness is stronger than successor fairness in the sense that for every Kripke structure K , if a path π of K is fair with respect to the transition

fairness condition, then π is also fair with respect to the successor fairness condition. The opposite is not necessarily true, as π may satisfy the successor fairness condition and not satisfy the transition fairness condition. However, it is not hard to see that given a Kripke structure with a successor fairness condition one can obtain, by inserting a new state in the middle of every transition, a corresponding Kripke structure for which each transition-fair path induces a corresponding successor-fair path in the original structure. The “stretching” effect the new states have on the paths can be easily overcome. Our results are thus also valid for the successor fairness condition.

In the rest of this paper we study the model-checking problem and the problem of finding dominators and postdominators for Kripke structures augmented with the transition fairness condition. For simplicity, we will only use the terms “fair path” or “fair Kripke structure”, without repeating the type of fairness.

2.3 Observations on fair paths

Recall that if a fair path π visits a state v infinitely often, then it also visits all the successors of v infinitely often. Such a fair behavior cascades from every successor to its own successors. As we formally state below, this guarantees that a fair path eventually gets trapped in some bottom MSCC, where it traverses all the states of the MSCC infinitely often. Formally, we have the following:

Lemma 1. *Let $\mathcal{K} = \langle K, \alpha \rangle$, with $K = \langle AP, W, R, W_0, L \rangle$, be a fair Kripke structure. If π is a fair path in \mathcal{K} then there exists a bottom MSCC C in K , and an index i , such that $\text{inf}(\pi) = \text{visit}(\pi^i) = C$.*

Lemma 2. *Let $\mathcal{K} = \langle K, \alpha \rangle$, with $K = \langle AP, W, R, W_0, L \rangle$, be a fair Kripke structure. Every (possibly empty) finite path w_0, w_1, \dots, w_k in \mathcal{K} can be extended to an infinite fair path $w_0, w_1, \dots, w_k, \dots$.*

3 Fair Model Checking

In this section we reduce fair model checking to non-fair CTL model checking, and analyze the complexity of the algorithm that follows. We start with CTL, and then proceed to fragments of LTL.

3.1 Fair CTL model checking

Our fair CTL model-checking algorithm is based on a function f that maps each CTL formula φ to another CTL formula $f(\varphi)$ such that for every fair Kripke structure $\mathcal{K} = \langle K, \alpha \rangle$ and state w in it, we have that \mathcal{K}, w satisfies φ fairly iff \mathcal{K}, w satisfies $f(\varphi)$ non-fairly. The function f is independent of K and the size of $f(\varphi)$ is linear in the size of φ (recall that we assume a DAG representation of CTL formulas. Thus, a sub-formula that appears more than once is represented by a single node, and is evaluated only once). Formally, we have the following:

Theorem 1. *There is a function $f : \text{CTL formulas} \rightarrow \text{CTL formulas}$ such that for every CTL formula φ , the following hold.*

- (1) $|f(\varphi)| = O(|\varphi|)$.
(2) For every fair Kripke structure $\mathcal{K} = \langle K, \alpha \rangle$, with $K = \langle AP, W, R, W_0, L \rangle$, and state $w \in W$, we have $\mathcal{K}, w \models_F \varphi$ iff $K, w \models f(\varphi)$.

Proof: We define f by induction on the structure of φ as follows.

- $f(\mathbf{true}) = \mathbf{true}$ and $f(\mathbf{false}) = \mathbf{false}$.
- $f(p) = p$ for $p \in AP$.
- $f(\neg\varphi_1) = \neg f(\varphi_1)$.
- $f(\varphi_1 \vee \varphi_2) = f(\varphi_1) \vee f(\varphi_2)$.
- $f(EX\varphi_1) = EXf(\varphi_1)$.
- $f(AX\varphi_1) = AXf(\varphi_1)$.
- $f(E\varphi_1 U \varphi_2) = Ef(\varphi_1)Uf(\varphi_2)$.
- $f(A\varphi_1 U \varphi_2) = A(f(\varphi_1) \wedge EFf(\varphi_2))Wf(\varphi_2)$.

It is easy to see that $|f(\varphi)| = O(|\varphi|)$. We prove Claim (2) in detail. The proof proceeds by an induction on the structure of φ . The induction base, where $\varphi = \mathbf{true}$, $\varphi = \mathbf{false}$, or $\varphi = p$, for $p \in AP$ is trivial. For the induction step, assume that Claim (2) holds for φ_1 and φ_2 . That is, for every fair Kripke structure \mathcal{K} , and every state $w \in W$, we have that $\mathcal{K}, w \models_F \varphi_1$ iff $K, w \models f(\varphi_1)$, and similarly for φ_2 .

The cases where φ is of the form $\neg\varphi_1$, $\varphi_1 \vee \varphi_2$, $EX\varphi_1$, $AX\varphi_1$, or $E\varphi_1 U \varphi_2$ are relatively easy, and are described in [ABK04]. Intuitively, the cases $EX\varphi_1$, $AX\varphi_1$, and $E\varphi_1 U \varphi_2$ all follow from the fact that their truth value is established by a finite prefix of a path which by Lemma 2 can always be extended to a fair path. Here we consider the more interesting case, where $\varphi = A\varphi_1 U \varphi_2$.

As an intuition, consider the special case where $\varphi = AFp$. Note that $f(AFp) = A(EFp)Wp$. Thus, we have to prove that all the fair paths that start at some state w eventually reach a state labeled p iff all (possibly unfair) paths that start at w either eventually reach p or always visit states from which a state labeled p is reachable. The ‘if’ direction is true since, by Lemma 1, every fair path has a tail for which all reachable states are actually visited. The ‘only if’ direction is true since, by Lemma 2, every state is the initial state of some fair path.

Assume first that $\mathcal{K}, w \models_F A\varphi_1 U \varphi_2$. Consider a (possibly unfair) path π starting at w . We show that either (case 1) there is $k \geq 0$ such that $\pi[k] \models f(\varphi_2)$ and all $0 \leq i < k$ are such that $\pi[i] \models f(\varphi_1) \wedge EFf(\varphi_2)$, or (case 2) all $i \geq 0$ are such that $\pi[i] \models f(\varphi_1) \wedge EFf(\varphi_2)$. It follows that $K, w \models A(f(\varphi_1) \wedge EFf(\varphi_2))Wf(\varphi_2)$.

We distinguish between two cases, which actually correspond to the two cases above. If there is a state along π that fairly satisfies φ_2 , we prove that case 1 above holds: let $k \geq 0$ be the minimal index for which $\pi[k] \models_F \varphi_2$. By Lemma 2, the path $\pi[0], \dots, \pi[k]$ can be extended to a fair path. Since $\mathcal{K}, w \models_F A\varphi_1 U \varphi_2$, our choice of k guarantees that $\pi[k] \models_F \varphi_2$ and $\pi[i] \models_F \varphi_1$ for all $0 \leq i < k$. By the induction hypothesis, $\pi[k] \models f(\varphi_2)$ and $\pi[i] \models f(\varphi_1)$ for all $0 \leq i < k$. Moreover, since $\pi[k] \models f(\varphi_2)$, then for all $0 \leq i < k$, we have that $\pi[i] \models EFf(\varphi_2)$. Thus, case 1 holds for π .

In the other case, that is, if no state along π fairly satisfies φ_2 , we prove that case 2 above holds. Consider an index $i \geq 0$. By Lemma 2, the path $\pi[0], \dots, \pi[i]$ can be extended to a fair path π_i . Since $\mathcal{K}, w \models_F A\varphi_1 U \varphi_2$, there is an index k such that

$\pi_i[k] \models_F \varphi_2$ and $\pi_i[j] \models_F \varphi_1$ for all $0 \leq j < k$. Since $\pi[j] \not\models_F \varphi_2$ for all $j \leq i$, it must be that $k > i$. By the induction hypothesis, $\pi_i[j] \models f(\varphi_1)$ for all $0 \leq j < k$. Moreover, since $\pi_i[k] \models f(\varphi_2)$, then for all $0 \leq j < k$, we have that $\pi_i[j] \models E F f(\varphi_2)$. Recall that $k > i$. Thus, in particular, $\pi_i[i] = \pi[i] \models f(\varphi_1) \wedge E F f(\varphi_2)$, and case 2 holds for π .

Assume now that $K, w \models A(f(\varphi_1) \wedge E F f(\varphi_2)) W f(\varphi_2)$. Let π be a fair path that starts at w . We first prove that there must be an index $k \geq 0$ such that $K, \pi[k] \models f(\varphi_2)$. since π is fair, By Lemma 1, there is a bottom MSCC $C \subseteq K$, and an index i , such that $visit(\pi^i) = C$. Assume by way of contradiction that no k as above exists. Then for all $j \geq 0$, we have that $K, \pi[j] \models (f(\varphi_1) \wedge E F f(\varphi_2))$. Since C is at the bottom, all states reachable from states in C are also in C . Hence, as $\pi[i] \in C$ and $K, \pi[i] \models E F f(\varphi_2)$, there is a state $v \in C$ such that $v \models f(\varphi_2)$. Since, however, $C = visit(\pi^i)$, there must be $k \geq 0$ such that $v = \pi[k]$, and we reach a contradiction. So, let k be the minimal index for which $K, \pi[k] \models f(\varphi_2)$. Then, by assumption, $K, \pi[i] \models f(\varphi_1)$ for all $0 \leq i \leq k$. Thus, by the induction hypothesis, $K, \pi \models \varphi_1 U \varphi_2$, and we are done. \square

For the sake of completeness, we state here the result of applying the function f on abbreviated CTL operators.

- $f(E F \varphi_1) = E F f(\varphi_1)$.
- $f(A F \varphi_1) = A(E F f(\varphi_1)) W f(\varphi_1)$.
- $f(E G \varphi_1) = E f(\varphi_1) U A G f(\varphi_1)$.
- $f(A G \varphi_1) = A G f(\varphi_1)$.
- $f(E \varphi_1 W \varphi_2) = E f(\varphi_1) U (f(\varphi_2) \vee A G f(\varphi_1))$.
- $f(A \varphi_1 W \varphi_2) = A f(\varphi_1) W f(\varphi_2)$.

Since f involves a linear blow-up and reduces fair CTL model checking to non-fair CTL model checking, the improved complexity for fair CTL model checking follows from the known complexity of the non-fair case [CES86,BCM⁺92]:

Corollary 1. *The fair CTL model-checking problem $\mathcal{K} \models \varphi$ for \mathcal{K} augmented with transition fairness can be solved in time linear in $|\mathcal{K}|$ and $|\varphi|$, or using at most $|\mathcal{K}||\varphi|$ symbolic steps.*

Interestingly, the function f maps *universal CTL* (ACTL, for short) formulas (that is, formulas that do not use the existential path quantifier, and whose satisfaction is preserved under simulation [GL94]) to CTL formulas that do use the existential path quantifier. As we now show, the use of the existential path quantifier is essential. We prove it already for the ACTL formula $A F p$.

Theorem 2. *There is no ACTL formula φ such that for all fair Kripke structures $\mathcal{K} = \langle K, \alpha \rangle$, we have $\mathcal{K} \models_F A F p$ iff $K \models \varphi$.*

Proof: Consider the two Kripke structures K and K' appearing in Figure 1 below. Note that K' is obtained from K by eliminating the state w_2 and the transitions to and from it. Since a path that consistently avoids w_2 in K is not fair, $K \models_F A F p$. On the other hand, since the single path of K' is fair and never reaches a state labeled p , we have that $K' \not\models A F p$. Accordingly, the formula φ we seek should be such that $K \models \varphi$

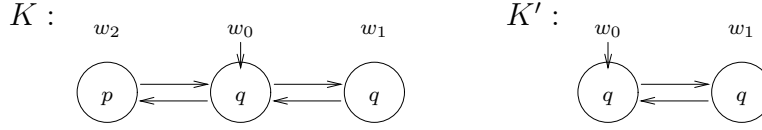


Fig. 1. While K simulates K' , only K fairly satisfies AFp .

and $K' \not\models \varphi$. Since, however, K' is simulated by K , and satisfaction of ACTL formulas is preserved under simulation [GL94], no such φ exists. \square

In Section 5 we discuss the theoretical aspects of Theorem 2, and its relation to the open problem about the expressive power of the linear fragments of CTL and ACTL.

3.2 Beyond CTL

The correctness of the function f follows from the observations on fair paths studied in Section 2.3. In this section we use these observations in the context of LTL model checking. We show that for many common LTL formulas, fair model checking can be reduced to fair CTL model checking, which in turn can be reduced to non-fair CTL model checking⁴. The LTL formulas we consider are those that specify acceptance by Büchi, Rabin, and Streett automata. Formally, we have the following.

Theorem 3. *Let $l_1, u_1, l_2, u_2, \dots, l_k, u_k$ be atomic propositions, for some $k \geq 1$. Consider the following three forms of LTL formulas:*

1. (Büchi) $\theta = GF l_1$.
2. (Rabin) $\theta = \bigvee_{1 \leq i \leq k} (GF l_i \wedge FG u_i)$.
3. (Streett) $\theta = \bigwedge_{1 \leq i \leq k} (GF u_i \vee FGl_i)$.

There is a partial mapping $g : \text{LTL formulas} \rightarrow \text{CTL formulas}$ such that for every LTL formula θ of one of the three forms above, the following hold.

- (1) $|g(\theta)| = O(|\theta|)$.
- (2) *For every fair Kripke structure $\mathcal{K} = \langle K, \alpha \rangle$, with $K = \langle AP, W, R, W_0, L \rangle$, and every state $w \in W$, there is a fair path π in \mathcal{K} , starting at w , such that $K, \pi \models \theta$ iff $K, w \models_F g(\theta)$.*

Proof: We define g as follows.

1. $g(GF l_1) = EGEF l_1$.
2. $g(\bigvee_{1 \leq i \leq k} (GF l_i \wedge FG u_i)) = EF(\bigvee_{1 \leq i \leq k} (AGEF l_i \wedge AG u_i))$.
3. $g(\bigwedge_{1 \leq i \leq k} (GF u_i \vee FGl_i)) = EF(\bigwedge_{1 \leq i \leq k} (AGEF u_i \vee AGL_i))$.

⁴ We note that, unlike other reductions of LTL to CTL model checking [KV98b, BRS99], our method here makes use of the fact that the LTL formulas should hold only in the fair paths of the Kripke structure.

It is easy to see that $|g(\theta)| = O(|\theta|)$. The proof of the correctness of g is given in [ABK04]. \square

One immediate application of Theorem 3 is fair LTL model checking of formulas of the form $\neg\theta$, for θ of one of the three forms above (note that Theorem 3 follows the existential approach, where one looks for a path that violates the property). A more ambitious application is to use the function g in order to fairly model check all LTL formulas: let ψ be an LTL formula. In the automata-theoretic approach to LTL model checking [VW94], we check whether all the paths of a Kripke structure K satisfy ψ by checking whether the product of K with an automaton $\mathcal{A}_{\neg\psi}$ that accepts all the words that violate ψ does not contain a *bad path* — a path whose projection on the states of $\mathcal{A}_{\neg\psi}$ is an accepting run of $\mathcal{A}_{\neg\psi}$. Indeed, the projection of such a path on the states of K is a path of K that is accepted by $\mathcal{A}_{\neg\psi}$, and thus violates ψ . The search for a bad path is reduced to checking whether there is a path in the product that satisfies an LTL formula θ induced by the acceptance condition of $\mathcal{A}_{\neg\psi}$. Thus, assuming $\mathcal{A}_{\neg\psi}$ is a Büchi, Rabin, or Streett automaton, θ is of one of the forms handled in Theorem 3. When the Kripke structure K is augmented with a fairness condition, ψ is violated iff the product contains a path whose projection on the states of $\mathcal{A}_{\neg\psi}$ is an accepting run of $\mathcal{A}_{\neg\psi}$, and whose projection on the states of K is a fair path of K . A detection of such a path can be done as described above by augmenting the product with a fairness condition. Then, the path that satisfies θ should be a fair path. To conclude, fair LTL model checking can be reduced to checking whether the product of K and $\mathcal{A}_{\neg\psi}$ has a fair path that satisfies an LTL formula of a specific form.

Unfortunately, the fact that K is augmented with the transition fairness condition does not imply that fair paths in the product correspond to fair paths in K , and vice versa. Technically, the product of K with $\mathcal{A}_{\neg\psi}$ may have non-fair paths whose projection on the states of K is a fair path of K . Below we show that the problem exists already for deterministic Büchi automata, implying that determinization, or a restriction to the Büchi acceptance condition do not solve this problem. (See [ABK04] for the proof.)

Theorem 4. *There is a fair Kripke structure $\mathcal{K} = \langle K, \alpha \rangle$ and a deterministic Büchi automaton \mathcal{A} with a set F of accepting states such that there exists a fair path in \mathcal{K} that is accepted by \mathcal{A} , yet the product of \mathcal{K} and \mathcal{A} has no fair path that visits infinitely many states whose projection on the states of \mathcal{A} is in F .*

Reducing fair LTL model checking to fair CTL model checking, we are able to use Corollary 1, and obtain similar complexity results for LTL:

Corollary 2. *The fair LTL model-checking problem $\mathcal{K} \models_F \neg\theta$ for \mathcal{K} augmented with transition fairness and θ of one of the forms handled in Theorem 3 can be solved in time linear in $|\mathcal{K}|$ and $|\theta|$, or using at most $|\mathcal{K}||\theta|$ symbolic steps. Moreover, if ψ is an LTL formula and $\mathcal{K} \models_F \psi$ iff the product of \mathcal{K} with $\mathcal{A}_{\neg\psi}$ fairly satisfies $\neg\theta$, then the LTL model-checking problem $\mathcal{K} \models_F \psi$ can be solved in time linear in $|\mathcal{K}|$ and $|\mathcal{A}_{\neg\psi}|$, or using at most $|\mathcal{K}||\mathcal{A}_{\neg\psi}|$ symbolic steps.*

We note that the automaton $\mathcal{A}_{\neg\psi}$ may be exponential in $|\psi|$ [VW94]. Moreover, sometimes we can achieve the desired property of the product having the transition fairness condition by applying to $\mathcal{A}_{\neg\psi}$ transformations that blow-up its state space further. Nevertheless, since K is typically much larger than ψ and the exponential blow up, as well as additional blow ups, do rarely appear in practice, the technique still pays off, as the algorithm that follows is linear in $|K|$, rather than quadratic in $|K|$, if one keeps the fairness condition (recall that the traditional algorithm for fair LTL model checking is quadratic in $|K|$).

4 Computation of Postdominators and Dominators

In this section we use the technique developed in Section 3 in order to describe an efficient and symbolic algorithm for calculating the dominators and post-dominators of states in a Kripke structure augmented with transition fairness.

For a state w of a Kripke structure K , the set of postdominators of w , denoted $pd(w)$, is the set of states s such that all paths from w eventually reach s . Dually, the set of dominators of w , denoted $dom(w)$, is the set of states s such that all paths to w pass through s . Let K^{rev} be the Kripke structure derived from K by reversing the direction of transitions. That is, K^{rev} is identical to K , only that its transition relation R^{rev} is such that for all states w and w' , we have that $R^{rev}(w, w')$ iff $R(w', w)$. It is not hard to see that $dom(w)$ is the set of states s such that all paths from w eventually reach s in K^{rev} . Thus, a state s is in $pd(w)$ in K iff s is in $dom(w)$ in K^{rev} . We study here the computation of postdominators. By the above, our results can be applied also for the computation of dominators.

As discussed in Section 1, the calculation of $pd(w)$ is not an easy problem. On the other hand, it is not hard to calculate, given s , all the states w for which $s \in pd(w)$. Indeed, w has to (fairly) satisfy AFs . Let $pd^{-1}(s)$ be the set of all states w such that s is in $pd(w)$. That is, $pd^{-1}(s) = \{w : s \in pd(w)\}$. The definition of pd^{-1} still leaves open the problem of calculating pd effectively. In many applications, however, the goal is to calculate all pairs $\langle s, w \rangle$ such that $s \in pd(w)$. Then, the fact that we consider pd^{-1} is not a disadvantage. Indeed, instead of calculating $pd(w)$ for all w , we can calculate $pd^{-1}(s)$ for all s . On the other hand, such an approach requires linearly many calculations of pd^{-1} – one calculation for each $s \in W$. So, even if calculation of pd^{-1} is reduced to a single fair CTL model-checking query, and even when such a query is reduced, as described in Section 3, to a non-fair model-checking query, and is calculated symbolically, we have to apply it linearly many times. The whole procedure is therefore not truly symbolic⁵.

We describe a truly symbolic algorithm for this task. Consider a Kripke structure $K = \langle AP, W, R, W_0, L \rangle$. Let $PD = \{\langle s, w \rangle : s \in pd(w)\}$, and let x_1, \dots, x_n be n variables used for encoding the state space of K . We assume that K is given symbolically. In particular, we are given an ROBDD f_R over $x_1, \dots, x_n, x'_1, \dots, x'_n$ that describes the transition relation R . We generate an ROBDD f_{pd} over $x_1, \dots, x_n, x'_1, \dots, x'_n$ such that $f_{pd}(x_1, \dots, x_n, x'_1, \dots, x'_n) = \mathbf{true}$ iff the states s and w encoded by x_1, \dots, x_n and x'_1, \dots, x'_n , respectively, are such that $\langle s, w \rangle \in PD$.

⁵ For a similar challenge, in the context of *coverage in model checking*, see [CKV01].

To explain our algorithm, we first describe it for a Kripke structure K with no fairness. Then, $PD = \{\langle s, w \rangle : w \models AFs\}$. Consider the operator (on a set $P \subseteq W \times W$) $PairAX(P) = \{\langle v, w \rangle : \text{for all successors } u \text{ of } w, \text{ we have } \langle v, u \rangle \in P\}$. Thus, for each pair $\langle v, w \rangle$ in P , the operator $PairAX(P)$ applies the AX operator (universal preimage) to w , leaving v unchanged. Note that $PairAX$ can be implemented symbolically: given an ROBDD for P and the ROBDD f_R , the construction of an ROBDD for $PairAX(P)$ is similar to the construction of an ROBDD for $AX(S)$, given an ROBDD for a set S of states [BCM⁺92]. Now, let $P_0 = \{\langle w, w \rangle : w \in W\}$, and $P_{i+1} = P_i \cup PairAX(P_i)$, for all $i \geq 0$. Intuitively, P_i contains all pairs $\langle s, w \rangle$ such that all the paths from w reach s within at most i transitions. It follows that the fixed-point of the above sequence is the set PD . In other words, we can describe PD by means of the fixed-point expression $\mu y. R_0 \vee PairAX(y)$, and calculate it symbolically.

When K is augmented by a transition fairness condition, $PD = \{\langle s, w \rangle : w \models_F AFs\}$. Equivalently, by Theorem 1, $PD = \{\langle s, w \rangle : w \models A(EFs)Ws\}$. Recall that $A\varphi_1 W\varphi_2 = \neg E(\neg\varphi_2)U(\neg\varphi_1 \wedge \neg\varphi_2)$. Accordingly, $A(EFs)Ws = \neg E(\neg s)U(AG\neg s)$. Consider the operator $PairEX(P) = \{\langle v, w \rangle : \text{there is a successor } u \text{ of } w \text{ for which } \langle v, u \rangle \in P\}$. Like $PairAX$, the operator $PairEX$ is “the pair version” of the operator EX , and it can be implemented symbolically. Finally, let $P_0 = \{\langle w, s \rangle : w \neq s\}$. Using P_0 , $PairAX$, and $PairEX$, we can describe the set PD by means of the (alternation free) fixed-point expression $\neg\mu y. \nu z. (R_0 \wedge AXz) \vee (P_0 \wedge PairEX(y))$, and calculate it symbolically.

We note that, in the context of postdominators and dominators, K is often augmented with a fairness condition that restricts attention to paths that visit some designated states. In particular, in $pd(w)$ we care for paths that start at w and eventually reach a halting or error states, and in $dom(w)$ we care for paths that start at some initial state and eventually reach w . Our idea above applies also in these cases. Then, instead of using the technique in Section 3, removal of fairness is easier. To see this, consider the problem of calculating $pd^{-1}(s)$ in a Kripke structure in which the fairness condition restricts attention to paths that eventually visit a state labeled *end*. Note that then, $w \models_F AFs$ iff $w \models \varphi$, for the CTL* formula $\varphi = A(End \rightarrow Fs)$. As proved in [EH86], φ has an equivalent CTL formula, and PD can be calculated using the operator $PairAX$.

5 Discussion

We studied reasoning about systems augmented with the transition fairness condition. We showed that while fairness usually makes reasoning harder, this is not the case for transition fairness.

The key to our results is the ability to translate a specification φ to a CTL formula φ' of size linear in the size of φ such that φ is satisfied fairly iff φ' is satisfied non-fairly. The formula φ' may contain the existential path quantifier E , even if φ imposes only universal requirements (that is, φ is in LTL or ACTL). For example, if $\varphi = AFp$, then $\varphi' = A(EFp)Wp$, and we showed that the use of the existential path quantifier in φ' is required – no φ' in ACTL can do the job. These observations are of interest with respect

to the relative expressive power of the linear fragments of CTL and ACTL. Consider an LTL formula ψ . Assume that ψ has an equivalent CTL formula. Can we then guarantee that ψ also has an equivalent ACTL formula? This seems very likely, as ψ imposes only linear, and hence universal, requirements. The question, however, is open, and the characterization in [Mai00], of $LTL \cap ACTL$, may not apply for $LTL \cap CTL$. The fact that the existential path quantifier is essential in the domain of the function f implies that, in the context of transition fairness, the answer to the question is negative: there are formulas in $LTL \cap ACTL$ that have a non-fair equivalence in CTL, but no non-fair equivalence in ACTL.

Another issue that is still open is extending the technique described in Section 3.2 to full LTL. As explained there, a straightforward application of the automata-theoretic approach does not work, as the transition fairness condition of the Kripke structure induces a different type of fairness condition in the product. We are searching for properties of the specification automaton, possibly properties relating the Kripke structure and the specification automaton, with which the technique can be applied in all LTL model-checking instances. We note that even when the transition to an automaton with such a property involves an additional exponential blow-up, the technique pays off, as it reduces the LTL model-checking task to an evaluation of a single fixed-point on the product, rather than a nested fixed-point, in the case a reduction to CTL model checking is impossible.

References

- [ABK04] B. Aminof, T. Ball, and O. Kupferman. Reasoning about systems with transition fairness. Technical Report MSR-TR-2004-89, September 2004.
- [BCM⁺92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *I&C*, 98(2):142–170, June 1992.
- [BGS00] R. Bloem, H.N. Gabow, and F. Somenzi. An algorithm for strongly connected component analysis in $n \log n$ symbolic steps. In *Formal Methods in Computer Aided Design*, LNCS 1954, pages 37–54, 2000.
- [BKAW98] A.L. Buchsbaum, H. Kaplan, A. Rogers, and J.R. Westbrook. A new, simpler linear-time dominators algorithm. *ACM TOPLAS.*, 20(6):1265–1296, 1998.
- [BRS99] R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of linear time logic properties. In *Proc. 11th CAV*, LNCS 1633, pages 222–235, 1999.
- [CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, LNCS 131, pages 52–71, 1981.
- [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.
- [C⁺91] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Trans. Program. Lang. Syst.*, 13(4):451–490, 1991.
- [CGP99] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
- [CKV01] H. Chockler, O. Kupferman, and M.Y. Vardi. Coverage metrics for temporal logic model checking. In *7th TACAS*, LNCS 2031, pages 528 – 542, 2001.
- [EH86] E.A. Emerson and J.Y. Halpern. ‘Sometimes’ and ‘not never’ revisited: on branching versus linear time. *Journal of the ACM*, 33(1):151–178, 1986.

- [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. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 16, pages 997–1072. Elsevier, MIT Press, 1990.
- [FOW87] J. Ferrante, K. Ottenstein, and J. Warren. The program dependence graph and its use in optimization. *ACM Trans. Program. Lang. Syst.*, 9(3):319–349, 1987.
- [Fra86] N. Francez. *Fairness*. Springer-Verlag, New York, 1986.
- [GL94] O. Grumberg and D.E. Long. Model checking and modular verification. *ACM Trans. on Programming Languages and Systems*, 16(3):843–871, 1994.
- [GPP03] R. Gentilini, C. Piazza, and A. Policriti. Computing strongly connected components in a linear number of symbolic steps. In *14th ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, Baltimore, Maryland, 2003.
- [Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
- [HT96] M. Henzinger and J.A. Telle. Faster algorithms for the nonemptiness of Streett automata and for communication protocol pruning. In *Proc. 5th Scandinavian Workshop on Algorithm Theory*, LNCS 1097, pages 10–20, 1996.
- [KG96] O. Kupferman and O. Grumberg. Buy one, get one free!!! *Journal of Logic and Computation*, 6(4):523–539, 1996.
- [KPR98] Y. Kesten, A. Pnueli, and L. Raviv. Algorithmic verification of linear temporal logic specifications. In *Proc. 25th ICALP*, LNCS 1443, pages 1–16, 1998.
- [KPSZ02] Y. Kesten, A. Pnueli, E. Shahar, and L. Zuck. Network invariant in action. In *Proc. 13th CONCUR*, LNCS 2421, pages 101–115, 2002.
- [KV98a] O. Kupferman and M.Y. Vardi. Verification of fair transition systems. *Chicago Journal of Theoretical Computer Science*, 1998(2), March 1998.
- [KV98b] O. Kupferman and M.Y. Vardi. Relating linear and branching model checking. In *Proc PROCOMET*, pages 304 - 326, Chapman & Hall, 1998.
- [LH00] Timo Latvala and Keijo Heljanko. Coping with strong fairness. *Fundamenta Informaticae*, 43(1–4):175–193, 2000.
- [LPS81] D. Lehman, A. Pnueli, and J. Stavi. Impartiality, justice, and fairness – the ethics of concurrent termination. In *Proc. 8th ICALP*, LNCS 115, pages 264–277, 1981.
- [LT79] T. Lengauer and R.E. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Prog. Lang. and Sys.*, 1(1):121–141, 1979.
- [Mai00] M. Maidl. *Using Model Checking for System Verification*. PhD thesis, Ludwig-Maximilians-Universität München, 2000.
- [MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, Berlin, January 1992.
- [QS81] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th International Symp. on Programming*, LNCS 137, pages 337–351, 1981.
- [QS83] J.P. Queille and J. Sifakis. Fairness and related properties in transition systems - A temporal logic to deal with fairness. In *Acta Informatica* 19:195-220, 1983.
- [Tho90] W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, pages 165–191, 1990.
- [Var85] M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proc. 26th FOCS*, pages 327–338, October 1985.
- [VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.