Verification Conditions for ω-Automata and Applications to Fairness

Nils Klarlund*

TR 90-1080
February 1990

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*Partially supported by a grant from the University of Aarhus, Denmark: NSF Grant CCR-88-06979; and the Thanks of the Scandinavia Foundation, Inc. The author's electronic address is klarlund@cs.cornell.edu.

Verification Conditions for ω-Automata and Applications to Fairness

Nils Klarlund[1]
Cornell University

Abstract We present sound and complete verification conditions for proving that a program satisfies a specification defined by a deterministic Rabin automaton. Our verification conditions yield a simple method for proving that a program terminates under general fairness constraints. As opposed to previous approaches, our method is syntax-independent and does not require recursive applications of proof rules. Moreover using a result by Safra, we obtain the first direct method for proving that a program satisfies a Büchi automaton specification. Finally, we show that our method generalizes two earlier methods.

1 Introduction

Automata on infinite strings (ω-automata) provide a simple and general setting for syntax-independent program specification and verification. A fundamental problem in this theory is to find verification methods for demonstrating that a program satisfies a specification when both are expressed as automata. *Direct methods* for solving this problem are based on *invariants* that define a correspondence between program automaton and specification automaton [AL88,AS87,AS89,Jon87,KS89,LT87,Sis89,Var87]. Formulated in terms of invariants, *verification conditions* (also known as *proof rules* or *proof obligations*) permit proofs to be carried out in a style similar to Hoare's logic [Hoa69].

In this paper, we significantly extend the scope of previous methods. Specifically, we present verification conditions for both deterministic Rabin

[1]Partially supported by a grant from the University of Aarhus, Denmark; NSF grant CCR 88-06979; and the Thanks to Scandinavia Foundation, Inc.

Author's electronic address is `klarlund@cs.cornell.edu`.

and nondeterministic Büchi automata. Our results yield a direct method for proving that a program terminates under general fairness constraints—something which was previously not possible in the context of automata-theory. The method is universally applicable as opposed to earlier syntax-dependent methods that required recursive applications of proof rules. We believe that our results provide a new and simple understanding of what it means for a program to terminate under a general fairness constraint.

2 Previous Work

Alpern and Schneider [AS87,AS89] used deterministic finite-state *Büchi automata* as a method of specification. They obtained an indirect method of verification for finite-state nondeterministic Büchi automata using the fact that any such automaton can in principle be expressed as a Boolean combination of deterministic Büchi automata, although there is no known direct conversion[2]. In [MP87], Manna and Pnueli gave verification conditions for $\forall$-*automata*, which are expressively equivalent to Büchi automata, although again there is no known direct conversion of a Büchi automaton into an $\forall$-automaton. Sistla [Sis87] considered deterministic automata with acceptance conditions given as temporal formulas on automaton states with the modalities $F^{\infty}(f)$ (infinitely often f) and $G^{\infty}(f)$ (almost always f). He showed that sound and complete verification conditions exist for automata that are in a special conjunctive normal form in which each conjunct is a particularly simple disjunction.

Conjunctions of automata acceptance conditions are easy to handle when there is a method for each conjunct: simply apply that method for each of the conjuncts. Verification with disjunctive normal forms is more difficult, and no direct method has to our knowledge been presented in the literature.

Other indirect methods, based on manipulations of formulas in the CTL* temporal logic, are applicable to a variety of finite-state automata [BCM+90, CDK89].

[2] A method yielding $O(16^{n^2})$ automata each having $O(16^{n^2})$ states can be deduced from [SVW87].

3 Outline of Results

The ultimate goal of this research is to extend previous techniques to infinite-state automata with acceptance conditions in disjunctive normal form (DNF). We present a verification method for specifications defined by infinite-state *deterministic Rabin automata*, whose acceptance condition is a restricted disjunctive normal form. Although arbitrary DNF acceptance conditions are not handled, our verification conditions already have important applications:

1. termination under general fairness constraints;

2. verification conditions for (nondeterministic) Büchi automata;

3. verification conditions for Rabin $\forall$-automata that are generalizations of the $\forall$-automata in [MP87];

4. simplification of the verification conditions in [AS89] for disjunctions of deterministic Büchi automata.

Application **2** relies on the recent elegant result by Safra [Saf88]. Application **1** has been the subject of much study [AO83,Fra86,FK84,GFMdR85,Mai89, LPS81,SdRG89]. To our knowledge, the only other automata-theoretic approach to this area is Vardi's paper [Var87], where abstract transformations on recursive automata are used to obtain an indirect method for termination under fairness. Another approach to termination under fairness is that of explicit schedulers. This approach is also not direct as it is based on manipulations of programs [DH86,Har86]. Other earlier methods suffered from being confined to programs with the inflexible syntax of nondeterministic loops containing sets of guarded commands. The proof rules involved recursive applications of themselves on syntactically transformed programs. It was suspected, however, that something much simpler was going on behind the syntax.

Our results provide an abstract and lucid automata-theoretic account of termination under general fairness constraints. We obtain a proof method that is simple in practice, because it depends neither on syntax nor on program transformations. This method is an immediate consequence of our main result, since general fairness constraints on nondeterministic programs are essentially acceptance conditions for deterministic Rabin automata.

4 Technical Development

Both programs and specifications are represented by automata. In earlier works [AS89,MP87], programs are represented as state transition systems. Instead we represent a program as a *looping automaton* A_Π, which is an infinite-state nondeterministic automaton $(\mathcal{E}, Q_\Pi, Q^0_\Pi, \rightarrow_\Pi)$, where $\mathcal{E}$ is the *alphabet* of the automaton (representing *events* of some sort); Q_Π is the set of *(program) states*; Q^0_Π is the set of *initial states*; and $\rightarrow_\Pi \subseteq Q_\Pi \times \mathcal{E} \times Q_\Pi$ is the *transition* or *rewriting relation*. The sets $\mathcal{E}$, Q_Π, Q^0_Π, $\rightarrow_\Pi$ are finite or countable.

A *run* $p_0, p_1, \ldots$ over $w = e_0, e_1, \ldots$ is a sequence of states such that $p_0 \xrightarrow{e_0}_\Pi p_1 \xrightarrow{e_1}_\Pi \cdots$ and $p_0 \in Q^0_\Pi$. A *partial run* is a finite prefix of a run. A word $w \in \mathcal{E}^\omega$ is *accepted* by A_Π if there is a run of A_Π over w. The *language* $L(A_\Pi)$ is the set of all words w such that there is a run of A_Π over w. A *reachable* state is a state that is contained in some partial run. We assume w.l.o.g. that program automata have no dead states (i.e. reachable states that are not in any run have been deleted).

A specification is represented as a *deterministic Rabin automata* A_Σ=

$$(\mathcal{E},\ Q_\Sigma,\ \rightarrow_\Sigma,\ s^0_\Sigma,\ \langle (L_0, U_0), \ldots, (L_N, U_N)\rangle),$$

where relation $\rightarrow_\Sigma$ is deterministic[3] and $s^0_\Sigma \in Q_\Sigma$ is the *initial state*. The list $\langle (L_0, U_0), \ldots, (L_N, U_N)\rangle$ is the Rabin *acceptance condition* consisting of *Rabin pairs* (L_χ, U_χ). Each $\chi \in [0..N]$ is the *color* of the Rabin pair (L_χ, U_χ). For technical reasons, we assume w.l.o.g. that $(L_0, U_0)=(\emptyset, \emptyset)$.

The Rabin pair (L_χ, U_χ) is *accepting* for the run $s_0, s_1, \ldots$ if for all H, there exists an $i > H$ such that $s_i \in L_\chi$, and there exists a K such that for all $i > K$, $s_i \notin U_\chi$. Intuitively, L_χ is a set of "good" states to be met infinitely often and U_χ is a set of "bad" states to be eventually avoided. An *accepting run* (of A_Σ) over w is a run $s_0, s_1, \ldots$ such that there is a color χ

[3]For each $s \in Q_\Sigma$, $e \in \mathcal{E}$, there is a most one $s' \in Q_\Sigma$ such that $s \xrightarrow{e}_\Sigma s'$.

for which (L_χ, U_χ) is accepting. The *language* $L(A_\Sigma)$ is the set of all words w such that there is an accepting run of A_Σ over w.

The disjunctive normal form corresponding to a Rabin acceptance condition is $\bigvee_{\chi \in [0..N]} L_\chi \wedge \neg U_\chi$.

5 Verification conditions

The verification conditions to be presented permit one to prove that $L(A_\Pi) \subseteq L(A_\Sigma)$, where $L(A_\Pi)$ is a looping automaton and $L(A_\Sigma)$ is a deterministic Rabin automaton. When $L(A_\Pi) \subseteq L(A_\Sigma)$, we say that program automaton A_Π *satisfies* specification automaton A_Σ.

The verification method is based on maintaining a correspondence between program states and specification automaton states augmented with "progress" information. In order to describe this correspondence, we need a few definitions.

An *indicator* δ is a pair (s, τ), where s is a specification state and τ is a *stack* of $n+1$ hypotheses $\langle \tau^0, \ldots, \tau^n \rangle$, with $n \leq N$. Define *size*(τ)=n. The *hypothesis* τ^ℓ, $\ell \leq n$, has the form (χ^ℓ, ν^ℓ). The *level* of hypothesis τ^ℓ is the number ℓ. Color $\chi^\ell \in [0..N]$ indicates that $(L_{\chi^\ell}, U_{\chi^\ell})$ is a candidate for an accepting pair. Ordinal ν^ℓ is the value of the *progress function* that measures progress towards reaching a good state of the hypothesis at level ℓ, i.e. towards a state in L_{χ^ℓ}.

The notion of *indicator rewriting* is the key to our results:

Definition 1 (Indicator Rewriting) *For indicators* $\delta = (s, \tau)$ *and* $\delta' = (s', \tau')$,

$$\delta \xrightarrow{e} \delta' \quad \textit{if}$$

$$(\delta 1) \qquad s \xrightarrow{e}_\Sigma s', \quad \textit{and}$$

$$(\delta 2) \qquad s, \tau \;\rightarrow\; s', \tau'.$$

Condition $(\delta 1)$ states that the specification automaton A_Σ can make a transition. In condition $(\delta 2)$, "$\rightarrow$" denotes *stack rewriting*, which ensures that "progress" is made. Stack rewriting can best be understood from the picture in Figure 1. Formally, it is defined as:

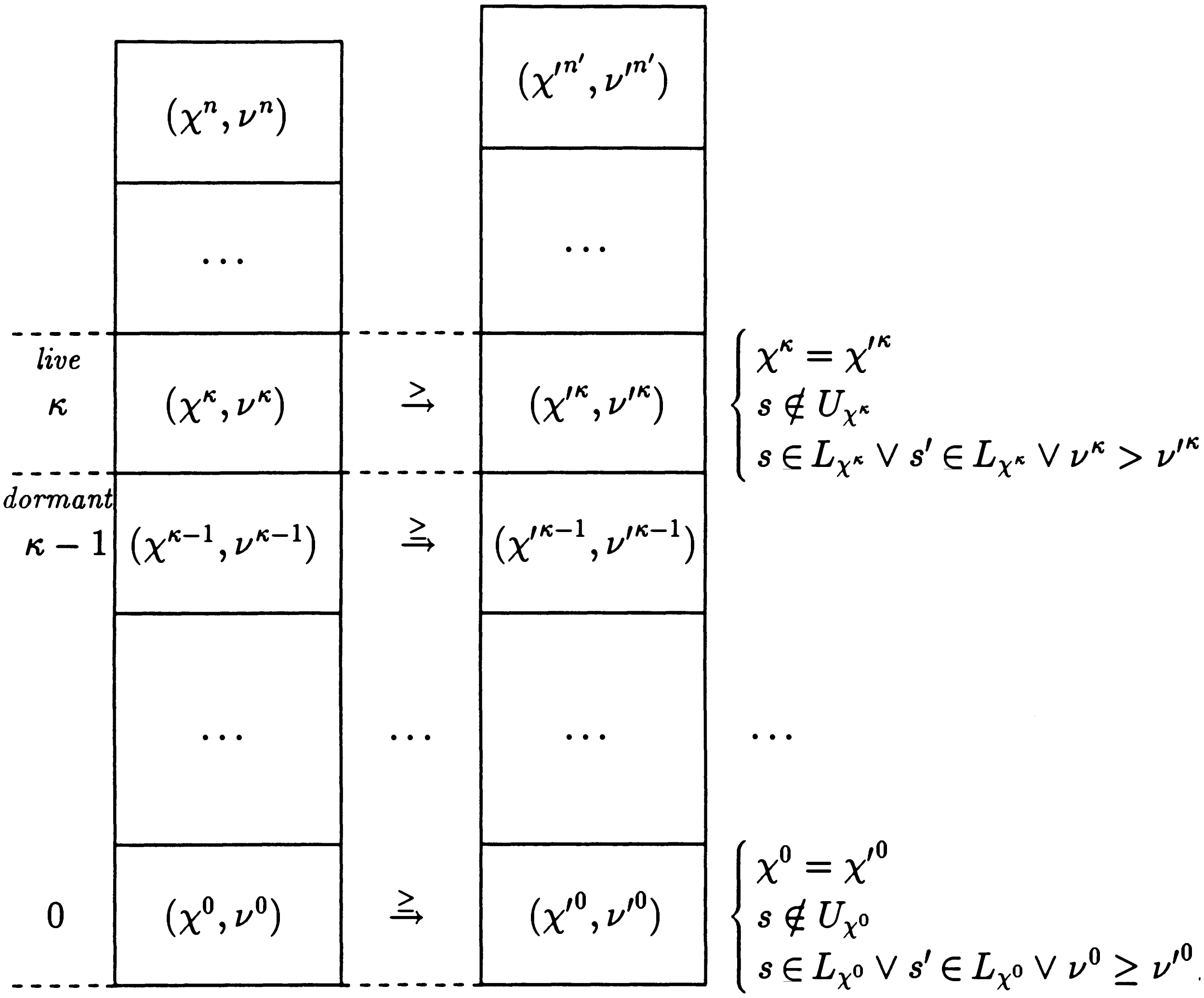

Figure 1: Stack rewriting $s, \tau \;\longrightarrow\; s', \tau'$.

Definition 2 (Stack Rewriting)

$$s, \tau \longrightarrow s', \tau' \;\; \textit{if} \;\; \exists \kappa \leq \min\{size(\tau), size(\tau')\} \;\; s.t$$

$(\tau 1)$ $\quad s, \nu^\kappa \xrightarrow{>} s', \nu'^\kappa$, *and*

$(\tau 2)$ $\quad \forall \lambda < \kappa: \; s, \tau^\lambda \xrightarrow{\geq} s', \tau'^\lambda$

Here, $\xrightarrow{>}$ and $\xrightarrow{\geq}$ are *hypothesis rewritings*. The κ in the definition is called the *height* of the stack rewriting. Informally, stack rewriting means that $(\tau 1)$ the hypothesis at level κ is "alive," i.e. the designated Rabin pair is getting closer towards becoming an accepting pair; whereas $(\tau 2)$ hypotheses underneath are "dormant." Formally,

Definition 3 (Live Hypothesis Rewriting)

$$s,(\chi,\nu) \stackrel{>}{\rightarrow} s',(\chi',\nu') \quad \textit{if}$$

(υ1) $\chi = \chi'$,

(υ2) $s \notin U_\chi$, *and*

(υ3>) $s \in L_\chi \;\vee\; s' \in L_\chi \;\vee\; \nu > \nu'$.

Definition 4 (Dormant Hypothesis Rewriting)

$$s,(\chi,\nu) \stackrel{\geq}{\Rightarrow} s',(\chi',\nu') \quad \textit{if } (\upsilon 1), (\upsilon 2), \textit{ and}$$

(υ3≥) $s \in L_\chi \;\vee\; s' \in L_\chi \;\vee\; \nu \geq \nu'$.

Under both rewritings, it is required that (υ1) the color of the hypotheses be unchanged and that (υ2) the old state s be not a bad state. In addition, live rewriting $\stackrel{>}{\rightarrow}$ requires that (υ3 >) either one of states s or s' be good, or that the ν, ν' ordinals measure headway towards getting to a good state; dormant rewriting $\stackrel{\geq}{\Rightarrow}$ requires that (υ3 ≥) either one of states s or s' be good, or that headway be not lost towards getting to a good state.

The correspondence between program automaton and specification automaton is expressed by an *invariant* relation $\mathcal{I}(p,\delta)$ that to each program state p associates a set of indicators δ.

We are now ready to state the verification conditions. They are:

(V1) $p \in Q^0_\Pi \;\Rightarrow\; \exists \delta : \mathcal{I}(p,\delta) \wedge \mathit{Init}(\delta)$

(V2) $p \xrightarrow{e}_\Pi p' \wedge \mathcal{I}(p,\delta) \;\Rightarrow\; \exists \delta' : \delta \xrightarrow{e} \delta' \wedge \mathcal{I}(p',\delta')$

Here, predicate $\mathit{Init}(\delta)$ of (V1) holds if $s = s^0_\Sigma$, where $\delta = (s,\tau)$. Condition (V1) simply states that any initial program state has an indicator containing the initial specification state. Condition (V2) ensures that when the program automaton performs a move on e, then any indicator δ associated with the old state p can be rewritten to an indicator δ' associated with the new state p'. Together, obligations (V1) and (V2) guarantee that if $p_0 \xrightarrow{e_0}_\Pi p_1 \xrightarrow{e_1}_\Pi \cdots$ is an accepting run of A_Π, then there is a sequence $\delta_0 \xrightarrow{e_0} \delta_1 \xrightarrow{e_1} \cdots$ such that $\mathit{Init}(\delta_0)$ holds. Thus, the verification conditions (V1) and (V2) resemble those that are used for other automata in [AS87,AS89,KS89,MP87,Sis89,Sta88]. As shown next, the sequence $\delta_0 \xrightarrow{e_0} \delta_1 \xrightarrow{e_1} \cdots$ produces a run of A_Σ and ensures that the run is accepting.

6 Soundness

Theorem 1 *If $\mathcal{I}$ is an invariant satisfying* (V1) *and* (V2), *then* $L(A_\Pi) \subseteq L(A_\Sigma)$.

Proof Let $p_0 \xrightarrow{e_0}_\Pi p_1 \xrightarrow{e_1}_\Pi \cdots$ be a run of A_Π. Using (V1) and (V2) we see there are indicators $\delta_i = (s_i, \tau_i)$ with

$$\tau_i = \langle(\chi_i^0, \nu_i^0), \ldots, (\chi_i^k, \nu_i^k)\rangle, \text{ where } k = size(\tau_i),$$

such that $\delta_0 \xrightarrow{e_0} \delta_1 \xrightarrow{e_1} \cdots$ and $Init(\delta_0)$. Hence, as for all i, $s_i \xrightarrow{e_i}_\Sigma s_{i+1}$ (by ($\delta 1$)) and $s_0 = s_\Sigma^0$ (by definition of *Init*), it follows that $s_0 \xrightarrow{e_0}_\Sigma s_1 \xrightarrow{e_1}_\Sigma \cdots$ is a run (the only one possible) of A_Σ over $e_0, e_1, \ldots$ We must prove that the run $s_0 \xrightarrow{e_0}_\Sigma s_1 \xrightarrow{e_1}_\Sigma \cdots$ is accepting.

By ($\delta 2$), for all i, the stack rewriting $s_i, \tau_i \rightarrow s_{i+1}, \tau_{i+1}$ holds; let κ_i be the associated rewriting height. Let $\kappa = \liminf_{i \to \infty} \kappa_i$. This limit exists since the rewriting heights are bounded by N.

By definition of κ, there is a K such that for all $i \geq K$, $\kappa_i \geq \kappa$. Thus by definition of stack rewriting, for all $i \geq K$, either $s_i, \tau_i^\kappa \xrightarrow{>} s_{i+1}, \tau_{i+1}^\kappa$ or $s_i, \tau_i^\kappa \xrightarrow{\geq} s_{i+1}, \tau_{i+1}^\kappa$. Then according to ($\upsilon 1$), there is a χ such that for all $i \geq K$, $\chi_i^\kappa = \chi$. We argue that (L_χ, U_χ) is an accepting pair for the run $s_0 \xrightarrow{e_0}_\Sigma s_1 \xrightarrow{e_1}_\Sigma \cdots$. By ($\upsilon 2$) and as $\kappa_i \geq \kappa$ for $i \geq K$, it holds for all $i \geq K$ that $s_i \notin U_\chi$.

The rest of this proof demonstrates that infinitely often, $s_i \in L_\chi$.

For a contradiction, suppose that there is a $K' \geq K$ such that for all $i \geq K'$, $s_i \notin L_\chi$. Then, as $\kappa_i \geq \kappa$ for $i \geq K'$, it follows from ($\upsilon 3 >$) and ($\upsilon 3 \geq$) that

(1) $\quad \nu_{K'}^\kappa \geq \nu_{K'+1}^\kappa \geq \nu_{K'+2}^\kappa \geq \cdots$

Using condition ($\tau 1$) of stack rewriting, we see that for infinitely many i, namely for all those i such that stack rewriting $s_i, \tau_i \rightarrow s_{i+1}, \tau_{i+1}$ has height κ,

$$s_i, \tau_i^\kappa \xrightarrow{>} s_{i+1}, \tau_{i+1}^\kappa.$$

Therefore, by ($\upsilon 3 >$), infinitely many "$\geq$" in (1) are strict. As the ν's are ordinals, we have arrived at a contradiction. □

7 Completeness

In the following $G = (V, E)$ denotes a directed graph with V and E countable. The graph G restricted to a set $W \subseteq V$ is denoted $G|W$. An *infinite path* is a set $v_0, v_1, \ldots$ of nodes in V such that for all $i \geq 0$, $(v_i, v_{i+1}) \in E$. Note that a single node v can define an infinite path if $(v, v) \in E$. An infinite path is *proper* if it is not eventually constant, i.e. it does not contain an infinite suffix $v, v, \ldots$

Lemma 1 *Let $G = (V, E)$ be a graph with no proper infinite paths. There is an assignment $\nu : V \to ORD$ of ordinals to V such that if $(v, v') \in E$ and $v \neq v'$, then $\nu(v) > \nu(v')$.*

Proof Let $G' = (V, E')$ be obtained from G by deleting all edges (v, v) from E. Then there are no infinite paths in G'. Hence, E' is a well-founded relation and this implies the existence of ν satisfying the lemma. □

Before stating a key lemma about colorable graphs, we need some definitions. A *color set assignment* CS is a map $V \to 2^{[0..N]}$. It associates a set of *permissible* colors $CS(v) \subseteq [0..N]$ to each node $v \in V$. An infinite path $v_0, v_1, \ldots$ is *eventually χ-permissible* if there is a K such that for all $i \geq K$, $\chi \in CS(v_i)$. A color set assignment is *eventually permissible* if every infinite path is eventually χ-permissible for some χ. A set $W \subseteq V$ is *χ-colorable* if for all $v \in W$, $\chi \in CS(v)$; in that case, χ is called a *valid* color of W. A set is *mono-colorable* if it is χ-colorable for some χ.

A *coloring* c of a set V is a mapping $c : V \to [0..N] \cup \{\bot\}$. If $c(v) = \bot$ then c does not assign a color to v, otherwise $c(v)$ is the color of v. A coloring c *obeys* a color set assignment CS if for all $v \in V$, $c(v) = \bot$ or $c(v) \in CS(v)$. An infinite path $v_0, v_1, \ldots$ is *eventually χ-stable*, where $\chi \in [0..N]$, if there is a K such that for all $i \geq K$, $\chi = c(v_i)$. A coloring c is *eventually stable* if every infinite path is eventually χ-stable for some χ. A set $W \subseteq V$ is *mono-colored* if there is $\chi \in [0..N]$ such that for all $v \in W$, $c(v) = \chi$.

The following graph-theoretic lemma is essential to the completeness proof.

Lemma 2 *Let $G = (V, E)$ be a graph. If CS is an eventually permissible color set assignment, then there is an eventually stable partial coloring $c : V \rightarrow [0..N] \cup \{\perp\}$ obeying CS. Furthermore, there is an assignment $\nu : V \rightarrow ORD$ of ordinals to V and an equivalence relation R on V, congruent with ν and c, such that each equivalence class is either mono-colored or* singular. *A singular class consists of one loop-free node v s.t. $c(v) = \perp$.*[4] *If $(v, v') \in E$ then either*

1. *vRv' (hence, $\nu(v) = \nu(v')$ and $c(v) = c(v') \neq \perp$), or*
2. *$\nu(v) > \nu(v')$.*

Proof We define an operator Γ, which is applied trans-finitely to yield an equivalence relation R on V and a coloring c obeying CS such that each equivalence class W of G/R is either mono-colored or singular. We will later prove that there is no proper infinite path in G/R.

Assuming now that G/R has no infinite paths, we prove that c is eventually stable. Define $h : G \rightarrow G/R$ to be the natural homomorphism. Let $c_R(W)$ be the image of c under h, i.e. $c_R(W) = \perp$ if W is singular, and $c_R(W)$ is the common color of nodes in W if W is mono-colored. It follows from the assumption that for any infinite path $P = v_0, v_1, \ldots$ in G, the path $h(P)$ in G/R consists of a finite set of classes and the last class W of the path must be a mono-colored class; it could not be a singular class because the node in such a class does not have a self-loop. Thus, the path P is eventually $c_R(W)$-stable because there is a K such that for all $k \geq K$, $v_k \in W$ and $c(v_k) = c_R(h(v_k)) = c_R(W)$.

To define an assignment ν of ordinals to V, we apply Lemma 1 to G/R, thereby obtaining an assignment $\nu_R : G/R \rightarrow ORD$. Define $\nu(v) = \nu_R(h(v))$. Now, let $(v, v') \in E$. Either vRv' or $h(v) \neq h(v')$. In the first case, $W = h(v) = h(v')$ is a mono-colored class. Hence, $c(v) = c(v') = c_R(W) \neq \perp$ and $\nu(v) = \nu(v') = \nu_R(W)$. In the second case, $(h(v), h(v')) \in E/R$; therefore, $\nu(v) = \nu_R(h(v)) > \nu(v') = \nu_R(h(v'))$ by Lemma 1. Hence, R and ν satisfies the properties stated in the lemma.

We now define Γ and prove that G/R has no proper infinite paths. Let $\Gamma : V \times V \rightarrow V \times V$ be

[4] A node v is *loop-free* if $(v, v) \notin E$.

$$(2)\quad \Gamma(S)=\begin{cases} S\cup\{(v',v'')\mid v',v''\in\mathcal{R}(v,(V\backslash S)^{\infty})\} & \text{where } (v,\chi)=\text{least}(v,\chi) \text{ s.t.}\\ & \mathcal{R}(v,(V\backslash S)^{\infty}) \text{ is } \chi\text{-colorable}\\ S & \text{if no such } (\nu,\chi) \text{ exists}\end{cases}$$

Here, the set $\mathcal{R}(v,W)$, where $v\in W\subseteq V$, is $\{v'\mid v\rightarrow^*_W v'\}$, the set of nodes in W *reachable* from v by a path in W;[5] the set W^{∞}, where $W\subseteq V$, is the set $\{u\in W\mid \text{some infinite path in } G|W \text{ originates in } u\}$; the notation $V|S$, where $S\subseteq V\times V$ is a relation, denotes $\{v\in V\mid vSv\}$, the set of nodes *marked* by S; and the complement of $V|S$, i.e. the set of *unmarked* nodes $\{v\in V|\neg(vSv)\}$, is denoted $V\backslash S$.

In (2), we have assumed a well-ordering on $V\times[0..N]$; it permits a new equivalence class to be determined uniquely by the least value (v,χ) defining a mono-colorable class.

The operator Γ is obviously a monotone set operator. Hence, there is an ordinal γ such that the set $S^*=\Gamma^{\gamma}$ is a least fixed point of Γ; here, Γ^{α} abbreviates $\Gamma^{\alpha}(\emptyset)$. The nodes $V|S^*$ are the set of *marked nodes* of V and the relation S^* is clearly an equivalence relation on $V|S^*$.

For reason of clarity, we omitted the definition of the coloring c in (2). When $\mathcal{R}(v,(V\backslash S)^{\infty})$ is added to S, where (v,χ) is the least value such that $\mathcal{R}(v,(V\backslash S)^{\infty})$ is χ-colorable, then we define $c(v')=\chi$ for all $v'\in\mathcal{R}(v,(V\backslash S)^{\infty})$. In this way, v is assigned a color $c(v)$ obeying $CS(v)$ for all marked v. For unmarked v, define $c(v)=\perp$.

In order to extend S^* to all V, define the equivalence relation R on V as vRv' if vS^*v' or $v=v'\in V\backslash S^*$. It is obvious that R is an equivalence relation on V and that marked nodes equivalent under S^* form a mono-colored class; the other classes are of the form $\{v\}$. Each such v is loop-free. This follows from the stronger statement that $V_{\infty}=(V\backslash S^*)^{\infty}$ is empty.

To prove that V_{∞} is empty, we suppose for a contradiction that u_0 is a node in V_{∞}. The following procedure yields a sequence of nodes $u_0,v_1,u_1,v_1,\ldots$ in V_{∞}.

If $\mathcal{R}(u_0,V_{\infty})$ is not 0-colorable, then there is some v_0 reachable in $G|V_{\infty}$ from u_0 such that $0\notin CS(v_0)$ (perhaps $v_0{=}u_0$). Let u_1 be a successor of v_0 (possibly v_0 itself).[6] A successor exists because there is an infinite path

[5] $v\rightarrow^*_W v'$ holds if there is some path $v_0,\ldots,v_n$ in $G|W$ with $v_0=v, v_n=v'$.

[6] A *successor* of a node u is a node v such that $(u,v)\in E$.

originating in each node of V_∞. If $\mathcal{R}(u_1, V_\infty)$ is not 1-colorable, then there is some node v_1 reachable from u_1 such that $1 \notin CS(v_1)$ (perhaps $v_1{=}u_1$).

Continuing in this way, we obtain either

- a node u_k such that $\mathcal{R}(u_k, V_\infty)$ is $(k \bmod (N+1))$-colorable, or
- an infinite path containing the nodes $u_0, v_0, u_1, v_1, \ldots$ such that each u_k is not $(k \bmod (N+1))$-colorable.

The second case would contradict the assumption that CS is eventually permissible. Therefore, there must be a $v \in V_\infty$ such that $\mathcal{R}(v, V_\infty)$ is mono-colorable. But this contradicts that S^* is a fixed point. Hence, V_∞ is empty. We conclude that there is no infinite path in G through singular classes; thus every infinite path in G/R contains infinitely many mono-colored classes.

To finish the proof, we need the following definition. For a mono-colored class W, let $\gamma(W)$ be the closure ordinal of W, i.e. the least ordinal α such that $W \times W \subseteq \Gamma^\alpha$. The ordinal $\gamma(W)$, which is a successor ordinal, indicates when the class W was marked. Note that before W was marked, W was contained in $(V \backslash S)^\infty$. Formally,

Claim 1 *If $W \in G/S^*$ and $\beta < \gamma(W)$, then $W \subseteq (V \backslash \Gamma^\beta)^\infty$.*

Proof If $v \in W \in G/S^*$, then $v \in (V \backslash \Gamma^{\gamma(W)-1})^\infty$, because W was added at step $\gamma(W)$. By the monotonicity of Γ, $v \in (V \backslash \Gamma^\beta)^\infty$ for all $\beta \leq \gamma(W) - 1$, i.e. for all $\beta < \gamma(W)$. □

To prove that there is no proper infinite path in G/R, we assume that $P_R = V_0, V_1, \ldots$ is such a path with $V_i \neq V_{i+1}$ for all i. We have just shown that infinitely many V_i are mono-colored. Let V_i and V_j $(i < j)$ be mono-colored classes such that for all k, $i < k < j$, V_k is a singular class $\{v_k\}$ (note that if $i = j + 1$ then there are no such singular classes).

Now suppose for a contradiction that $\gamma_i = \gamma(V_i) \leq \gamma(V_j) = \gamma_j$. Let $v_i \in V_i$, $v_j \in V_j$ such that $v_i, v_{i+1}, \ldots, v_j$ is a path in G. By the claim, $v_j \in (V \backslash \Gamma^{\gamma_i - 1})^\infty$ because $\gamma_i - 1 < \gamma_i \leq \gamma_j$. By the monotonicity of Γ and by assumption that $v_k \in V \backslash S^*$ for $i < k < j$, nodes $v_{i+1}, \ldots, v_{j-1}$ are in $V \backslash \Gamma^{\gamma_i - 1}$. Thus, nodes $v_i, v_{i+1}, \ldots, v_{j-1}$ are in $(V \backslash \Gamma^{\gamma_i - 1})^\infty$ because v_j is in $(V \backslash \Gamma^{\gamma_i - 1})^\infty$. Hence, by definition of Γ, all of V_j is included in V_i and we conclude that $V_i = V_j$. It follows that if $i = j+1$, then $V_i = V_{i+1}$, which we assumed not to be the case; and if $i < j+1$, then v_{i+1} should be in V_i. This contradicts that V_{i+1} is a singular class. Hence, $\gamma(V_i) > \gamma(V_j)$.

Let $t(i)$ be the monotone function that selects the indices of the infinite subsequence of all mono-colored classes of $V_0, V_1, \ldots$ We have that for all i, $\gamma(V_{t(i)}) > \gamma(V_{t(i+1)})$. This contradicts the well-foundedness of the ordinals. Thus, a proper infinite path $P_R = V_0, V_1, \ldots$ does not exist in G/R. □

Theorem 2 *If* $L(A_\Pi) \subseteq L(A_\Sigma)$, *then there is an invariant* $\mathcal{I}$ *satisfying* (V1) *and* (V2).

Proof The invariant will be obtained from the *joint graph* $G = (V, E)$. The nodes V, also called *joint states*, are the jointly reachable program and specification states. Formally,[7]

$$V = \{(p,s) \in Q_\Pi \times Q_\Sigma \mid \exists u \in \mathcal{E}^*,\ \exists p^0_\Pi \in Q^0_\Pi \text{ s.t. } p^0_\Pi \xrightarrow{u}_\Pi p \text{ and } s^0_\Sigma \xrightarrow{u}_\Sigma s\}.$$

Edges E are defined by the relation $((p,s),(p',s')) \in E$ if $p \xrightarrow{e}_\Pi p'$ and $s \xrightarrow{e}_\Sigma s'$. Note that any infinite path in G starting in a joint state (p^0_Π, s^0_Σ), where $p^0_\Pi \in Q^0_\Pi$, defines a run of both A_Π and A_Σ over some infinite word w. Such a path is called a *run* in G. We assume that $L(A_\Pi) \subseteq L(A_\Sigma)$. Therefore, every run in G defines an accepting run of A_Σ.

The algorithm **Assign** described below is used to associate an indicator $\delta(p,s) = (s, \tau(p,s))$ to each joint state (p,s). The invariant is defined as

$$\mathcal{I}(p,\delta) \text{ iff } \exists s : \delta = \delta(p,s).$$

That is, the indicators associated with program state p are those associated with joint states (p,s) of G.

Claim 2 *To verify* (V2) *it suffices to show that*

$$(3) \qquad \forall((p,s),(p',s')) \in E \ :\ s,\ \tau(p,s) \rightarrow s',\ \tau(p',s').$$

[7]For a transition relation $\rightarrow$ and a word $u \in \mathcal{E}^*$, $q \xrightarrow{u} q'$ holds if for some e there are $e_0, \ldots, e_n$ and $q_0, \ldots, q_{n+1}$ such that $u = e_0, \ldots, e_n$ and such that $q_0 = q$, $q_{n+1} = q'$ and $q_0 \xrightarrow{e_0} \cdots \xrightarrow{e_n} q_{n+1}$.

$\texttt{Assign}(W, X, \chi, \kappa)$:

1. Let $\widehat{W} = W \backslash (Q_\Pi \times L_\chi)$.
   Let $\widehat{X} = [0..N] \backslash (X + \chi)$.
2. Use Lemma 2 on $G|\widehat{W}$ with color set assignment $CS \cap \widehat{X}$ to obtain a partial coloring $c : \widehat{W} \to \widehat{X} + \perp$ obeying $CS \cap \widehat{X}$, an assignment ν of ordinals to $\widehat{W}$ and a relation R on $\widehat{W}$.
3. For all v in W:
   (a) Let $\chi^\kappa(v) = \chi$.
   (b) Let $\nu^\kappa(v) = \begin{cases} \nu(v) & \text{if } v \in \widehat{W} \\ 0 & \text{if } v \in W \backslash \widehat{W} \end{cases}$.
4. For each colored class $\widetilde{W}$ of $(G|\widehat{W})/R$:
   $\texttt{Assign}(\widetilde{W}, X + \chi, c_R(\widetilde{W}), \kappa + 1)$

Figure 2: The algorithm Assign.

Proof If $p \xrightarrow{e}_\Pi p'$ and $\mathcal{I}(p, \delta)$, then by definition of $\mathcal{I}$ there exists s such $\delta = \delta(p, s)$ and if s' is the state such that $s \xrightarrow{e}_\Sigma s'$, then $((p, s), (p', s')) \in E$.[8] Then by choosing $\delta' = \delta(p', s')$, it is clearly the case that $\mathcal{I}(p', \delta')$ holds and that condition ($\delta 1$) of indicator rewriting is satisfied. Therefore, it suffices to show that condition ($\delta 2$), i.e. $s, \tau(p, s) \to s', \tau(p', s')$, holds in order to establish $\delta \xrightarrow{e} \delta'$. □

The stacks $\tau(p, s)$ are obtained by applying the algorithm Assign in Figure 2 with initial parameters $(V, \emptyset, 0, 0)$. The application $\texttt{Assign}(V, \emptyset, 0, 0)$

[8]Here, we have used the assumption that A_Π has no dead states as follows. Let $u = e_0, \ldots, e_{i-1}$ be such that there exists a partial run $p_0 \xrightarrow{e_0}_\Pi \ldots p_{i-1} \xrightarrow{e_{i-1}}_\Pi p$ and such that $s^0_\Sigma \xrightarrow{u}_\Sigma s$ (such a u exists because (p, s) is joint reachable). By the assumption that A_Π has no dead states and as program state p' is a reachable state, state p' is contained in some run of A_Π. Hence, there are $p_{i+1}, \ldots$ and $e_{i+1}, \ldots$ such that $p' \xrightarrow{e_{i+1}}_\Pi p_{i+1} \xrightarrow{e_{i+2}}_\Pi \ldots$ It follows that $p_0 \xrightarrow{e_0}_\Pi \ldots p_{i-1} \xrightarrow{e_{i-1}}_\Pi p \xrightarrow{e}_\Pi p' \xrightarrow{e_{i+1}}_\Pi p_{i+1} \ldots$ is a run of A_Π. Hence, the word $e_0, \ldots, e_{i-1}, e, e_{i+1}, \ldots$ is in $L(A_\Pi)$. As $L(A_\Pi) \subseteq L(A_\Sigma)$, the word $e_0, \ldots, e_{i-1}, e, e_{i+1}, \ldots$ is accepted by A_Σ. This establishes the existence of s'.

and each subsequent application of `Assign` in Step 4 is called an *invocation*. The color set assignment CS used by `Assign` is defined as

$$CS(p,s) = \{\chi \mid s \notin U_\chi\}.$$

In other words, $CS(p,s)$ is the set of colors that can be used in an indicator without violating the constraint $(v2)$ of hypothesis rewriting.

The purpose of $\texttt{Assign}(W, X, \chi, \kappa)$ is to assign the color χ and ordinals to hypotheses at level κ of W, and to assign colors from $[0..N]\backslash(X+\chi)$ and ordinals to hypotheses at levels greater than κ such that rewriting of height at least κ is possible within all of W. Here, X denotes the set of colors that have already been assigned to levels less than κ.

In Step 1 of `Assign`, $\widehat{W} = W \backslash (Q_\Pi \times L_\chi)$ is the set of joint states in W that are not "good" with respect to color χ. $\widehat{X} = [0..N]\backslash(X+\chi)$ is the set of colors that may be used to color $\widehat{W}$ at higher levels. Here, the notation $X+\chi$ means $X \cup \{\chi\}$.

In Step 2, Lemma 2 is used to obtain a partial coloring c of $\widehat{W}$ obeying $CS \cap \widehat{X}$. Here, $CS \cap \widehat{X}$ denotes the colors in CS that are also in $\widehat{X}$, i.e. $(CS \cap \widehat{X})(v) = CS(v) \cap X$.

In Step 3, the color of hypotheses at level κ of $v \in W$ is defined to be χ and the ordinals assigned are those of Step 2.

Finally in Step 4, hypotheses at levels greater than κ are defined for each colored class $\widetilde{W}$ of $(G|\widehat{W})/R$.

For each joint node $v = (p,s)$, the size of the stack $\tau(u)$ is determined as the maximal value of κ for which $\chi^\kappa(u)$ is assigned a value. Note that as at level $\kappa+1$, `Assign` is invoked on disjoint subsets of level κ, $\chi^\kappa(u)$ is assigned a value by at most one invocation of `Assign`. To explain the algorithm more formally and to prove that Lemma 2 can indeed always be used in Step 2, we need some more terminology.

We say that a subset $W \subseteq V$ is *χ-tolerant* if for all $(p,s) \in U$, $s \notin U_\chi$, and that W is *χ-avoiding* if for all $(p,s) \in U$, $s \notin L_\chi$. Note, that as $L(A_\Pi) \subseteq L(A_\Sigma)$, for any run in G, there is a χ such that every suffix of the run is not χ-avoiding and there is a suffix that is χ-tolerant. A subset $W \subseteq V$ is *X-tolerant* if it is χ-tolerant for each χ in X. Similarly, a subset $W \subseteq V$ is *X-avoiding* if it is χ-avoiding for each χ in X.

Claim 3 *For each invocation* $\texttt{Assign}(W, X, \chi, \kappa)$*, the following holds:* $|X| = \kappa$*;* $\chi \notin X$*; and* W *is* $(X+\chi)$*-tolerant and* X*-avoiding.*

Proof (By induction) This is clearly true for the first invocation $\texttt{Assign}(V, \emptyset, 0, 0)$ because $X = \emptyset$; $\chi = 0$; $L_0 = U_0 = \emptyset$; and $\kappa = 0$.

When $\texttt{Assign}(W, X+\chi, c_R(W), \kappa+1)$ is applied from within $\texttt{Assign}$, it may by induction hypothesis be assumed that $|X| = \kappa$; $\chi \notin X$; and that W is $(X+\chi)$-tolerant and X-avoiding.

It follows that $|X+\chi| = \kappa+1$. Also, by definition of the color set assignment in Step 2, $c_R(\widetilde{W}) \notin X+\chi$. Further, as $\widetilde{W}$ is $c_R(\widetilde{W})$-tolerant by definition of the color set assignment, it is true that $\widetilde{W}$ is $(X+\chi+c_R(\widetilde{W}))$-tolerant. Finally, as $\widetilde{W}$ is χ-avoiding (since $\widetilde{W} \subseteq \widehat{W} = W \backslash (Q_\Pi \times L_\chi)$), it follows that $\widetilde{W}$ is $(X+\chi)$-avoiding. □

Consider an invocation $\texttt{Assign}(W, X, \chi, \kappa)$. By Claim 3, it follows that $\widehat{W}$ defined in Step 1 of $\texttt{Assign}$ is $(X+\chi)$-tolerant and $(X+\chi)$-avoiding. Let now P be a infinite path in $\widehat{W}$. It is $(X+\chi)$-avoiding because $\widehat{W}$ is $(X+\chi)$-avoiding. Hence, by the assumption that every run in G is accepting, P is accepting for a Rabin pair $(L_{\chi'}, U_{\chi'})$, where $\chi' \in \widehat{X} = [0..N] \backslash (X+\chi)$. In particular, P is eventually χ'-permissible. Therefore, $CS \cap \widehat{X}$ is an eventually permissible color set assignment of $\widehat{W}$ and Lemma 2 is applicable in Step 2 of $\texttt{Assign}$.

Now to show that all stacks have height at most N, we note that if $\kappa = N$, then as $|X| = \kappa$ and $\chi \notin X$ (by Claim 3), $X+\chi = [0..N]$. Hence in that case, the color set assignment of Step 2 defines for each node the set of permissible colors to be the empty set; thus, $G|\widehat{W}$ has no infinite paths and $\texttt{Assign}$ is not applied in Step 4.

We can now prove (3) of Claim 2. Assume that $(v, v') \in E$, where $v = (p, s)$ and $v' = (p', s')$. Let κ, the rewriting height, be the maximal level such that there are $W^0 \supseteq \cdots \supseteq W^\kappa$; $X^0, \ldots, X^\kappa$; and $\chi^0, \ldots, \chi^\kappa$ such that $W^0 = V$; $v, v' \in W^\kappa$; $X^0 = \emptyset$; $\chi^0 = 0$; and for all $\lambda \leq \kappa$, $\texttt{Assign}(W^\lambda, X^\lambda, \chi^\lambda, \lambda)$ is an invocation. The number κ exists, because $\texttt{Assign}(V, \emptyset, 0, 0)$ is an invocation. Also, all values $W^\lambda, X^\lambda, \chi^\lambda$ are unique.

From the definition of $\texttt{Assign}$ it follows that for all $\lambda \leq \kappa$:

$$\chi^\lambda(v) = \chi^\lambda(v') = \chi^\lambda.$$

Besides,

$$s \notin U_{\chi^\lambda(v)},$$

because W^λ is $\chi^\lambda(v)$-tolerant. Thus, for $\lambda \leq \kappa$, $(v1)$ and $(v2)$ of hypothesis rewriting are satisfied.

Furthermore, for $\lambda < \kappa$, $W^{\lambda+1}$ is an equivalence class of the relation R defined over $\widehat{W}^\lambda = W^\lambda \backslash (Q_\Pi \times L_{\chi^\lambda})$. Hence, as both v and v' are in the same equivalence class $W^{\lambda+1}$, it can be seen from Lemma 2 that $\nu^\lambda(v) = \nu^\lambda(v')$. Thus $(v3 \geq)$ is established and it it follows that dormant hypothesis rewriting

$$s, (\chi^\lambda(v), \nu^\lambda(v)) \overset{\geq}{\Rightarrow} s', (\chi^\lambda(v'), \nu^\lambda(v'))$$

holds for $\lambda < \kappa$. Hence, we have established $(\tau 2)$ of stack rewriting.

Finally, to see that $(\tau 1)$,

$$s, (\chi^\kappa(v), \nu^\kappa(v)) \overset{\geq}{\rightarrow} s', (\chi^\kappa(v'), \nu^\kappa(v')),$$

is valid, we now only need to demonstrate that $(v3 >)$ holds, i.e. $s \in L_{\chi^\kappa}$ or $s' \in L_{\chi^\kappa}$ or $\nu^\kappa(v) > \nu^\kappa(v')$. So consider the case where $s, s' \notin L_{\chi^\kappa}$. Then $v, v' \in \widehat{W}^\kappa = W^\kappa \backslash (Q_\Pi \times L_{\chi^\kappa})$. Thus, Lemma 2 was used to assign ordinals at level κ to v and v'. Hence, either $vR^\kappa v'$ or $\nu^\kappa(v) > \nu^\kappa(v')$. But if $vR^\kappa v'$, then κ would not have been maximal. Thus, $\nu(v) > \nu(v')$. □

Observations

It follows from the completeness proof that it is not necessary to assume that $(L_0, U_0) = (\emptyset, \emptyset)$ in the list of Rabin pairs if the list contains a pair of the form $(L_\chi, \emptyset)$. Such a pair can be put at the bottom of all stacks instead of $(L_0, U_0) = (\emptyset, \emptyset)$.

The formulation of hypothesis rewriting could have been made a little more strict without loss of soundness or completeness. If the clause "$s \in L_\chi$" is deleted from the definition of hypothesis rewritings $\overset{\geq}{\rightarrow}$ and $\overset{\geq}{\Rightarrow}$, the verification conditions remain obviously sound. That they remain complete follows from assigning ordinals to $W \backslash \widehat{W}$ in Step 3 of the algorithm according to:

Lemma 3 *Let $G = (V, E)$ be a graph, $V' \subseteq V$ and $\nu' : V' \rightarrow ORD$ an assignment of ordinals to V'. Then there is an extension $\nu : V \rightarrow ORD$ of ν' such that if $(u, v) \in E$, $v \notin V', v \in V$, then $\nu(u) > \nu(v)$.*

Proof For $v \notin V'$, de ne $\nu(u) = 1 + \sup\{\nu'(v) \mid v \in V'\}$. □

Similarly, one could instead remove the clause "$s' \in L_\chi$" from the definition of hypothesis rewritings.

8 Application 1: Fairness

Our technique allows automata-theoretic proofs of termination under a general fairness constraint $F = \{(\phi_1, \psi_1), \ldots, (\phi_N, \psi_N)\}$, which is defined in [FK84] or [Fra86, p. 112]. Each (ϕ_χ, ψ_χ), $1 \leq \chi \leq N$, is an *unfairness condition*, which consists of *enabling condition* ϕ_χ and *action condition* ψ_χ, both of which are program state predicates. A *computation* is an infinite sequence of program states. A computation is *unfair w.r.t.* (ϕ_χ, ψ_χ) if enabling condition ϕ_χ is satisfied infinitely often and action condition ψ_χ is satisfied only finitely often. (Thus, a computation is *fair w.r.t.* (ϕ_χ, ψ_χ) if enabling condition ϕ_χ is satisfied infinitely often implies that action condition ψ_χ is satisfied infinitely often.) A computation is *unfair* if it is unfair for some (ϕ_χ, ψ_χ). (Thus, a fair computation is one which is fair w.r.t. each (ϕ_χ, ψ_χ).)

Observe that a computation is unfair if and only if it satisfies F viewed as a Rabin acceptance condition.

A program Π *terminates under* F if every non-terminating computation of Π is unfair, i.e. if every non-terminating computation satisfies acceptance condition F.

In order to prove that Π terminates under F, we assume that Π is given by an initial program state p^0 and a nondeterministic relation $p \rightarrow p'$, which denotes that an atomic action can transform the program state from p to p'. Then Π can be represented as a deterministic automaton $A_\Pi = (Q_\Pi, Q_\Pi, p^0, \rightarrow_\Pi)$ by letting the alphabet $\mathcal{E}$ be the set of program states Q_Π and letting transitions be $p \xrightarrow{p'}_\Pi p'$, where $p \rightarrow p'$.

The specification automaton is the same as the program automaton—except for the acceptance condition, which is F augmented with a pair (ϕ_t, false) expressing the *termination* condition that a final state is entered:

$$\langle(\phi_1, \psi_1), \ldots, (\phi_n, \psi_n), (\phi_t, \mathit{false})\rangle.$$

Here, it is assumed that the final state is repeated infinitely often once it has been entered. This avoids dealing with finite computations.

Further, as program and specification automata only differ in the acceptance condition, it is only necessary to associate stacks and not indicators with program states. Also, according to the observations after Theorem 2, the termination condition can be kept at the bottom of the stack (and no pair $(\emptyset, \emptyset)$ is needed).

The verification conditions then become:

$$(\mathrm{V1}_F) \quad \exists \tau : \mathcal{I}(p^0, \tau)$$

$$(\mathrm{V2}_F) \quad p \to p' \ \wedge\ \mathcal{I}(p, \tau) \ \Rightarrow\ \exists \tau' : \tau, p \ \to\ \tau', p' \ \wedge\ \mathcal{I}(p', \tau').$$

Now, Theorems 1 and 2 yield the following characterization of programs that terminate under general fairness constraints:

Corollary 1 *A program* Π *terminates under general fairness constraint* F *if and only if there is an invariant* $\mathcal{I}(p, \tau)$ *satisfying* $(\mathrm{V1}_F)$ *and* $(\mathrm{V2}_F)$*. Further, it is only necessary to associate one stack with each program state, i.e.* $\mathcal{I}$ *needs only to be a function.*

Proof To prove the second assertion, observe that joint graph G in the proof of Theorem 2 have states of the form (p, p). □

This characterization implies that termination under general fairness can be demonstrated by using simple assertions about stacks.

Example

Program Π_{ex} shown in Figure 3 is taken from [GFMdR85] (and can also be found in [Fra86]). It is presented in the syntax of guarded statements [Dij76]. Program Π_{ex} terminates under assumption of *strong fairness*. This means that for any non-terminating computation there is some guarded statement ℓ that is unfairly executed, i.e. ℓ is infinitely often enabled but only executed finitely many times.

An informal account of why Π_{ex} terminates is as follows. In the beginning—while $x = 0$ holds—if guarded statement ℓ_a is continuously executed, then guarded statement ℓ_b will be enabled every second iteration but

```
ℓs: x,y:=0,0;
    *[ ℓa: x=0                →  y:=y+1
       ℓb: x=0 ∧ even(y)      →  x:=1
       ℓc: x≠0 ∧ y≠0          →  y:=y-1
       ℓd: x≠0 ∧ y≠0          →  skip]
ℓt: goto ℓt
```

Figure 3: The program Π_{ex}.

never executed, resulting in a computation that is unfair w.r.t. ℓ_b. Hence, for a fair computation ℓ_b is eventually executed, resulting in progress for the (literally) underlying termination hypothesis. Further, if ℓ_d is continuously executed, then guarded statement ℓ_c will be enabled infinitely often, resulting in an unfair computation w.r.t. ℓ_c. Hence, for a fair computation ℓ_c is executed until the loop is exited and the terminal state ℓ_t is entered.

The line of reasoning above is reflected in the formal argument, which is based on the use of stack assertions. A *stack assertion* at label ℓ has the form

$$\{P_1 \to \tau_1, \ldots, P_n \to \tau_n\},$$

where the P_i's are program predicates and the τ_i's are stack descriptors. A *stack descriptor* is a list of hypotheses that are functions of the program state. A *program state* has the form $(\ell, \vec{x})$ where ℓ is the value of the program counter (abbreviated PC) and $\vec{x}$ is a value assignment determining the values of the program variables. The stacks associated with a program state $(\ell, \vec{x})$ are the values of those stack descriptors τ_i evaluated at $(\ell, \vec{x})$ for which P_i is satisfied at $(\ell, \vec{x})$.

In Figure 4, program Π_{ex} is shown with stack assertions. Here, the Rabin pair with index

b means "ℓ_b is executed unfairly," i.e. $(\phi_b, \psi_b) = ((x = 0 \wedge even(y)), at\ell_b)$,

c means "ℓ_c is executed unfairly," i.e. $(\phi_c, \psi_c) = ((x \neq 0 \wedge y \neq 0), at\ell_c)$, and

$$
\begin{array}{l}
\{\langle(t,\omega+1)\rangle\} \\
\ell_s\texttt{: x,y:=0,0;} \\
\quad \{inv:\ x=0 \qquad\qquad \rightarrow \langle(t,\omega),\ (b,0)\rangle, \\
\qquad\qquad x=1 \wedge y\neq 0 \ \rightarrow \langle(t,y),\ (c,0)\rangle\} \\
\quad \texttt{*[}\ \ell_a\texttt{: x=0} \qquad\qquad\qquad \rightarrow \texttt{y:=y+1} \\
\qquad \ell_b\texttt{: x=0} \wedge even\texttt{(y)} \quad \rightarrow \texttt{x:=1} \\
\qquad \ell_c\texttt{: x}\neq\texttt{0} \wedge \texttt{y}\neq\texttt{0} \qquad \rightarrow \texttt{y:=y-1} \\
\qquad \ell_d\texttt{: x}\neq\texttt{0} \wedge \texttt{y}\neq\texttt{0} \qquad \rightarrow \textbf{skip}\texttt{]} \\
\quad \{\langle(t,0)\rangle\} \\
\ell_t\texttt{: } \textbf{goto } \ell_t
\end{array}
$$

Figure 4: The program Π_{ex} with assertions.

t means "Π_{ex} is in a terminal state," i.e. $(\phi_t, \psi_t) = (at\ell_t, false)$.

We will prove that the program satisfies the acceptance condition

$$\langle(\phi_b, \psi_b), (\phi_c, \psi_c), (\phi_t, false)\rangle,$$

i.e. either the guarded statement ℓ_b or ℓ_c is executed unfairly or the program terminates by reaching ℓ_t. Notice, that whether ℓ_a and ℓ_b are fairly executed has no importance for the termination of Π_{ex}.

An operational explanation of the stack assertions in Figure 4 is a follows. At the bottom of every stack resides the hypothesis t that the program terminates.

Consider the statement ℓ_s. Its execution results in progress for hypothesis t, because the ordinal of this hypothesis decreases from $\omega + 1$ to ω.

However, when $x = 0$ holds and statement ℓ_a is executed, hypothesis t is only "dormant." Instead, hypothesis b is on top of the stack to measure progress towards fulfillment of the unfairness condition for ℓ_b.

When x becomes 1, the termination hypothesis t is active and this allows hypothesis c—that the program executes unfairly with respect to ℓ_c—to be stacked. Thus, progress can take place for hypothesis c when ℓ_d is executed.

For a more formal argument, it is assumed that the initial state of Π_{ex} is $(\ell_s, \langle\perp,\perp\rangle)$, i.e. PC is ℓ_s and **x** and **y** are both undefined. For this program state, the stack $\langle(t, \omega+1)\rangle$ is associated—thus establishing ($\mathrm{V1}_F$). Upon execution of the statement at ℓ_s, the program state becomes $(\ell_a, \langle 0,0\rangle)$ or $(\ell_b, \langle 0,0\rangle)$ because only the guards at ℓ_a and ℓ_b are satisfied. The PC is at the loop as long as the label of the program state is ℓ_a, ℓ_b, ℓ_c or ℓ_d. The PC becomes ℓ_t when all guards are false, that is, when $x \neq 0 \wedge y = 0$.

To prove ($\mathrm{V2}_F$), we consider each statement at a time: for every stack τ possible according to the assertion at the statement (the precondition), there must be a stack τ' possible according to the assertion after the statement (the postcondition) such that $\tau \rightarrow \tau'$. In the following, values of the program variables, hypotheses and stacks after the execution are denoted by primed variable names.

For the statement at ℓ_s, there is only one stack possible, namely $\langle(t, \omega+1)\rangle$. After execution of ℓ_s : `x,y:=0,0`, the stack $\langle(t,\omega),(b,0)\rangle$ is possible, and

$$\langle(t, \omega+1)\rangle \rightarrow \langle(t,\omega),(b,0)\rangle$$

with rewriting height 0. The proof of this is immediate:

0: $(t, \omega+1) \overset{\geq}{\rightarrow} (t,\omega)$ because $\neg\psi_t = \mathit{true}$ and $\omega + 1 > \omega$,

where "0" denotes the level of the hypothesis rewriting. Statement ℓ_t is also easy. When ℓ_t is executed, there is only one stack possible, namely $\langle(t,0)\rangle$. After execution, the stack $\langle(t',0)\rangle$ is possible. But $\langle(t,0)\rangle \rightarrow \langle(t,0)\rangle$ as

0: $(t,0) \overset{\geq}{\rightarrow} (t,0)$ because $\neg\psi_t = \mathit{true}$ and $\phi_t = \mathit{at}\ell_t$ holds.

For the other statements, ℓ_a, ℓ_b, ℓ_c and ℓ_d, proofs are given below. The stack τ_α abbreviates $\langle(t,\omega),(b,0)\rangle$ and τ_β abbreviates $\langle(t,y),(c,0)\rangle$ from the loop invariant.

ℓ_a: $x = 0$ and τ_α is the only possible stack. After execution of ℓ_a, the PC will still be at the loop and stack τ_α is possible. Then, $\tau_\alpha \rightarrow \tau'_\alpha$ with height 1:

1: $(b,0) \overset{\geq}{\rightarrow} (b,0)$ because $\neg\psi_b = \neg \mathit{at}_b$ holds and if y is even, then $\phi_b = (x = 0 \wedge \mathit{even}(y))$ holds, otherwise y is odd and $\phi'_b = (x' = 0 \wedge \mathit{even}(y'))$ holds because $y' = y+1$ is even.

0: $(t,\omega) \xrightarrow{\geq} (t,\omega)$ because $\neg\psi_t = \mathit{true}$.

ℓ_b: $x = 0 \wedge \mathit{even}(y)$ and τ_α is the only possible stack. Two cases:

$y = 0$: Next PC is ℓ_t which allows τ_t'. Then, $\tau_\alpha \to \tau_t'$ as

0: $\langle(t,\omega)\rangle \xrightarrow{\geq} \langle(t,0)\rangle$ because $\psi_t = \mathit{false}$ and $\omega > 0$.

$y > 0$: Next PC is at loop and stack τ_β' is possible. Hence, $\tau_\alpha \to \tau_\beta'$ with height 0:

0: $(t,\omega) \xrightarrow{\geq} (t,y')$ because $\neg\psi_t = \mathit{true}$ and $\omega > y' = y$.

ℓ_c: $x \neq 0 \wedge y > 0$ and τ_β is the only possible stack. Two cases:

$y = 1$: Next PC is ℓ_t. Prove that $\tau_\beta \to \tau_t'$ with height 0:

0: $\langle(t,\omega)\rangle \xrightarrow{\geq} \langle(t,0)\rangle$ because $\psi_t = \mathit{false}$ and $\omega > 0$.

$y > 1$: Next PC is at loop and stack τ_β is possible. Prove $\tau_\beta \to \tau_\beta'$ with height 0.

0: $(t,y) \xrightarrow{\geq} (t,y')$ because $\neg\psi_t = \mathit{true}$ and $y > y' = y - 1$.

ℓ_d: $x \neq 0 \wedge y > 0$ and τ_β is only possibly stack. After transition τ_β' is possible stack. Prove $\tau_\beta \to \tau_\beta'$ with height 1.

1: $(c,0) \xrightarrow{\geq} (c,1)$ because $\neg\psi_c = \neg at_c$ and $\phi_c' = (x \neq 0 \wedge y \neq 0)$.

0: $(t,y) \xrightarrow{\geq} (t,y')$ because $\neg\psi_t = \mathit{true}$ and $y = y'$.

The termination proof in [Fra86,GFMdR85] of the simple program Π_{ex} is complicated, involving not only the original program, but also two transformed programs.

9 Application 2: Büchi automata

A finite-state Büchi automaton A is a tuple $(\mathcal{E}, Q, Q^0, \to, Q^F)$, where $(\mathcal{E}, Q, Q^0, \to)$ is a finite-state nondeterministic looping automaton. The set $Q^F \subseteq Q$ is the set of *accepting states*. A run $q_0, q_1, \ldots$ of A over a word w is accepting if there is a state in Q^F that occurs infinitely often in $q_0, q_1, \ldots$

Using Safra's result [Saf88], we define a direct method of verification with finite-state Büchi automata. Safra showed how to construct a deterministic

Rabin automaton $\mathcal{R}(A)$ with $O(2^{n \log n})$ states and $O(n)$ pairs given a non-deterministic Büchi automaton with n states. When applying our method of verification to $\mathcal{R}(A)$, we see that an indicator (r, τ) for the (determinized) Büchi automaton A contains a state r of $\mathcal{R}(A)$ and a stack τ with guesses of accepting pairs of $\mathcal{R}(A)$.

Corollary 2 *The verification conditions* (V1) *and* (V2) *applied to* $\mathcal{R}(A)$ *are sound and complete for verifying that a program satisfies a finite-state Büchi automaton* A *with* n *states. Each indicator* δ *contains a tree of at most* n *subsets of states of* A *and a stack of height at most* n.

Note that each indicator contains an amount of information essentially exponential in the size of the specification automaton. This is probably optimal because determinizations of a nondeterministic automata imply an exponential blow-up.

10 Application 3: Rabin $\forall$-automata

A Rabin $\forall$-automaton A is defined as the deterministic Rabin automaton in Section 4 except that the transition relation need not be deterministic and that there may be more than one initial state. A word $w \in \mathcal{E}^\omega$ is accepted by A iff *all* runs of A over w are accepting. The verification conditions (V1) and (V2) are changed to

V1$_\forall$: $p \in Q^0_\Pi \ \wedge \ s \in Q^0_\Sigma \Rightarrow \ \exists \tau : \mathcal{I}(p, s, \tau)$

V2$_\forall$: $p \to_\Pi p' \ \wedge \ s \to_\Sigma s' \ \wedge \ \mathcal{I}(p, s, \tau) \ \Rightarrow \ \exists \tau' : s, \tau \to s', \tau' \ \wedge \ \mathcal{I}(p', s', \tau')$

where the invariant relation $\mathcal{I}(p, s, \tau)$ associates a set of stacks to each (p, s). By slightly changing the proof of Theorem 1, this formulation of the verification conditions is seen to ensure that all runs of A over some word w are accepting.

Theorem 3 *If* $\mathcal{I}$ *is an invariant satisfying* (V1$_\forall$) *and* (V2$_\forall$), *then* $L(A_\Pi) \subseteq L(A_\Sigma)$.

Proof It is sufficient to prove that if $e_0, e_1, \ldots$ is any sequence of events such that there is a run $p_0 \xrightarrow{e_0} p_1 \xrightarrow{e_1} \cdots$ of A_Π, then every run $s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} \cdots$ of A_Σ is accepting. So consider a run $p_0 \xrightarrow{e_0} p_1 \xrightarrow{e_1} \cdots$ and a run $s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} \cdots$.

Using (V1) and (V2) we see there are stacks τ_i with

$$\tau_i = \langle (\chi_i^0, \nu_i^0), \ldots, (\chi_i^k, \nu_i^k) \rangle, \text{ where } k = \mathit{size}(\tau_i),$$

such that $s_0, \tau_0 \xrightarrow{e_0} s_1, \tau_1 \xrightarrow{e_1} \cdots$.

Now the arguments from the proof Theorem 1 can be repeated to show that there is an accepting pair for $s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} \cdots$ □

Theorem 4 *If $L(A_\Pi) \subseteq L(A_\Sigma)$, then there is an invariant $\mathcal{I}$ satisfying* ($\mathrm{V1}_\forall$) *and* ($\mathrm{V2}_\forall$).

Proof As in the proof of Theorem 2, the invariant will be obtained from the joint graph $G = (V, E)$. If $L(A_\Pi) \subseteq L(A_\Sigma)$, then every run in G defines a run of A_Π and an accepting run of A_Σ. It follows that exactly the same construction as in the proof Theorem 2 can be carried out. Define $\mathcal{I}(p, s, \tau)$ iff $\tau = \tau(p, s)$. Details left to the reader. □

From the completeness proof, it can be seen that the invariant relation can be restricted to be a function that associates *one* stack with each (p, s). The verification conditions ($\mathrm{V1}_\forall$) and ($\mathrm{V2}_\forall$) generalize those of [MP87] which dealt with $\forall$-automata having the acceptance condition $\langle (L, \emptyset), (Q_\Sigma, Q_\Sigma \backslash U) \rangle$ (so that a run is accepting if a state from L occurs infinitely often, or states of the run are eventually contained in U).

11 Application 4: Disjunctions of Deterministic Büchi automata.

We present an improvement of the method by Alpern and Schneider in [AS89] for demonstrating that a program satisfies a *disjunction*

$$\mathcal{D} = A^1 \vee \cdots \vee A^p \vee \neg A^{p+1} \vee \cdots \vee \neg A^{p+n}$$

of deterministic Büchi automata $A^i = (\mathcal{E}, Q, q^{i^0}, \rightarrow^i, Q^{i^F})$. Automata $A^1, \ldots, A^p$ are called *positive* automata and $A^{p+1}, \ldots, A^{p+n}$ are called *negative* automata. As in [AS89] we assume that the automata have no dead states.

The disjunction $\mathcal{D}$ defines the language $\mathcal{L}(\mathcal{D})$, which is

$$\mathcal{L}(A^1) \cup \cdots \cup \mathcal{L}(A^p) \cup \overline{\mathcal{L}(A^{p+1})} \cup \cdots \cup \overline{\mathcal{L}(A^{p+n})}$$

Stated in our terminology, the method in [AS89] relied on indicators each containing a state of every A^h for $1 \leq h \leq p+n$ together with a *candidate set* $C \subseteq [p+1,\ldots,n+p]$ and a progress function ν. The purpose of the candidate set was to identify automata that might never again enter accepting states.

Using our main result, we here prove that it is not necessary to identify a set of such automata—pointing at one candidate is sufficient. Our simpler verification conditions can still be written as (V1) and (V2), where automaton A_Σ is taken to be a product automaton of the A^i's. Hence, state space Q_Σ is $Q^1 \times \cdots \times Q^{p+n}$ and initial state q^0_Σ is $\langle {q^1}^0,\ldots,{q^{p+n}}^0\rangle$. Transition relation $\rightarrow_\Sigma$ is defined in the natural way as the product of transition relations $\rightarrow^i$. If we define projection function $\Pi^h s = s^h$, where $s = \langle s^1,\ldots,s^h,\ldots,s^{p+n}\rangle$, then the acceptance condition for A_Σ is $\langle (L,\emptyset),(Q_\Sigma,U^{p+1}),\ldots,(Q_\Sigma,U^{p+n})\rangle$, where $U^{p+k} = (\Pi^{p+k})^{-1}({Q^{p+k}}^F)$ (for $1 \leq k \leq n$) and $L = \{s \in Q_\Sigma \mid \exists h \in [1..p] \text{ s.t. } \Pi^h s \in {Q^h}^F\}$. It is trivial to see that $\mathcal{L}(Q_\Sigma) = \mathcal{L}(\mathcal{D})$.

An indicator now is a tuple (s,c,ν), where $s \in Q^\Sigma$, $c \in [p+1..p+n]$ points to the candidate, and ν is an ordinal. The predicate $Init(\delta)$ is true if $s = s^0_\Sigma$, where $\delta = (s,c,\nu)$. Indicator rewriting becomes

Definition 5 (Disjunctive Automata Indicator Rewriting) *For indicators $\delta = (s,c,\nu)$ and $\delta' = (s',c',\nu')$,*

$$\delta \xrightarrow{e} \delta' \quad \textit{if}$$

$(\delta 1_\mathcal{D})$ $\quad s \xrightarrow{e}_\Sigma s'$, *and*

$(\delta 2_\mathcal{D})$ $\quad (\exists h{:}1 \leq h \leq p{:}\, \Pi^h s \in {Q^h}^F) \vee (\nu > \nu') \vee (\nu \geq \nu' \wedge c = c' \wedge \Pi^c s \notin {Q^c}^F)$

Condition $(\delta 1_\mathcal{D})$ is the same as $\forall h : \Pi^h s \xrightarrow{e}{}^h \Pi^h s'$. Condition $(\delta 2_\mathcal{D})$ ensures that either some positive automaton enters an accepting state; or the variant function decreases; or it does not increase and the candidate c remains the same and negative automaton A^c is not in an accepting state.

Theorem 5 *With the definition of indicator rewriting above, (V1) and (V2) are sound and complete for proving that $L(A_\Pi) \subseteq L(\mathcal{D})$.*

Proof (Soundness) Let $\delta_0,\delta_1,\ldots$ be the sequence of indicators generated by a word $e_0,e_1,\ldots \in L(A_\Pi)$, where $\delta_i = (s_i,c_i,\nu_i)$. Then $s_0,s_1,\ldots$ is a run of A_Σ and $\Pi^h s_0, \Pi^h s_1, \ldots$ is a run of each A^h over $e_0, e_1, \ldots$ If for infinitely many i, $(\exists h : 1 \leq h \leq p : \Pi^h s_i \in Q^{h^F})$, then it can be seen that some positive machine accepts $e_0, e_1, \ldots$ Hence, in that case $e_0, e_1, \ldots \in \mathcal{L}(\mathcal{D})$.

Otherwise, there is a K, ν and c such that for all $i \geq K$, $\nu_i = \nu$, $c_i = c$, and $\Pi^c s_i \notin Q^{c^F}$. Hence, $e_0, e_1, \ldots \notin \mathcal{L}(A^c)$ and it follows that $e_0, e_1, \ldots \in \mathcal{L}(\mathcal{D})$.

(Completeness) Apply the completeness proof but modify it so that the pair $(L, \emptyset)$ is always at the bottom of the stack. This can be done because the set of "bad" states in $(L, \emptyset)$ is empty. Also note that we can discard all hypotheses at levels greater than 1 and that a progress function at level 1 is not needed. This follows from the fact that for any hypothesis at level 1, all states are "good" states. For any indicator $\widehat{\delta} = (s, \langle(0, \nu), (c, \nu')\rangle)$ of the modified completenes proof, form the indicator (s, ν, c). For any indicator $\widehat{\delta} = (s, \langle(0, \nu)\rangle)$ of the modified completeness proof, form the indicator $(s, \nu, p+1)$. Here, $p+1$ could instead have been any other negative automaton. Now, it is easy to see that $\widehat{\delta} \xrightarrow{e} \widehat{\delta'}$ implies $\delta \xrightarrow{e} \delta'$ if δ, δ' have been made from $\widehat{\delta}, \widehat{\delta'}$. □

12 Conclusion

Using an automata-theoretic approach, we have obtained:

- a direct verification method for Rabin-automata that generalizes methods in [AS89,MP87];
- a direct method of verification for specifications defined by Büchi automata;
- a simple method for proving termination under a general fairness constraint, which consists of a set of unfairness conditions. The method employs stacks of hypotheses, with the underlying termination hypothesis at the bottom. The other hypotheses each correspond to the fulfillment of an unfairness condition. The essence of our results is:
  A program terminates under general fairness if and only if for each program state the unfairness conditions can be ordered such that for any

program transition, progress towards fulfillment is made for one hypothesis, while the ones below are dormant.

The ideas behind our handling of disjunctions can be extended to general finite DNFs. We suspect, however, that our simple stack method cannot be extended to termination under extreme fairness, which is expressible as an infinite list of Rabin pairs.

Acknowledgments

Thanks to Dexter Kozen and Steven Mitchell for suggestions and advice.

References

[AL88] Martín Abadi and Leslie Lamport. The existence of refinement mappings. In *Proc. 2. Symp. on Logic in Computer Science.* IEEE, 1988.

[AO83] K.R. Apt and E.-R. Olderog. Proof rules and transformations dealing with fairness. *Science of Computer Programming*, 3:65–100, 1983.

[AS87] B. Alpern and F.B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2:117–126, 1987.

[AS89] B. Alpern and F.B. Schneider. Verifying temporal properties without temporal logic. *ACM Transactions on Programming Languages*, 11(1):147–167, January 1989.

[BCM+90] J.R. Burch, E.M. Clarke, K.L McMillan, D.L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proc. Symp. on Logic in Computer Science*, 1990.

[CDK89] 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. Technical report, Carnegie Mellon University, 1989.

[DH86] I. Dayan and D. Harel. Fair termination with cruel schedulers. *Fundamenta Informatica*, 9:1–12, 1986.

[Dij76] E.W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[FK84] N. Francez and D. Kozen. Generalized fair termination. In *Proc. 11th POPL, Salt Lake City.* ACM, January 1984.

[Fra86] Nissam Francez. *Fairness.* Springer-Verlag, 1986.

[GFMdR85] O. Grumberg, N. Frances, J.A. Makowsky, and W.P. de Roever. A proof rule for fair termination of guarded commands. *Information and Control*, 66(1/2):83–102, 1985.

[Har86] D. Harel. Effective transformations on infinite trees with applications to high undecidability, dominos, and fairness. *Journal of the ACM*, 33(1):224–248, 1986.

[Hoa69] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, October 1969.

[Jon87] B. Jonsson. Modular verification of asynchronous networks. In *Proc. Sixth Symp. on the Principles of Distributed Computing*, pages 152–166. ACM, 1987.

[KS89] N. Klarlund and F.B. Schneider. Verifying safety properties using infinite-state automata. Technical Report TR-1036, Cornell University, 1989.

[LPS81] D. Lehmann, A. Pnueli, and J. Stavi. Impartiality, justice and fairness: the ethics of concurrent termination. In *Proc. 8th ICALP, LNCS 115*. Springer-Verlag, 1981.

[LT87] N. Lynch and M. Tuttle. Hierarchical correctness proof for distributed algorithms. In *Proc. Sixth Symp. on the Principles of Distributed Computing*, pages 137–151. ACM, 1987.

[Mai89] M.G. Main. Complete proof rules for strong fairness and strong extreme-fairness. Technical Report CU-CS-447-89, Department of Computer Science, University of Colorado, 1989.

[MP87] Z. Manna and A. Pnueli. Specification and verification of concurrent programs by $\forall$-automata. In *Proc. Fourteenth Symp. on the Principles of Programming Languages*, pages 1–12. ACM, 1987.

[Saf88] S. Safra. On complexity of ω-automata. In *Proc. Foundations of Computer Science*. IEEE, 1988.

[SdRG89] F.A. Stomp, W.P. de Roever, and R.T. Gerth. The μ-calculus as an assertion-language for fairness arguments. *Information and Computation*, 82:278–322, 1989.

[Sis87] A.P. Sistla. On using automata in the verification of concurrent programs. Technical report, Computer and Intelligent Systems Laboratory, GTE Laboratories Inc, 1987.

[Sis89] A.P. Sistla. A complete proof system for proving correctness of nondeterministic safety specifications. Technical report, Computer and Intelligent Systems Laboratory, GTE Laboratories Inc., 1989.

[Sta88] E. Stark. Proving entailment between conceptual state specifications. *Theoretical Computer Science*, 56:135–154, 1988.

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

[Var87] M. Vardi. Verification of concurrent programs: The automata-theoretic framework. In *Proc. Symp. on Logic in Computer Science*. IEEE, 1987.