Annals of Pure and Applied Logic 51 (1991) 79–98
North-Holland

Verification of concurrent programs: the automata-theoretic framework*

Moshe Y. Vardi**

IBM Almaden Research Center, K53-802, 650 Harry Road, San Jose, CA 95120-6099, USA

Communicated by A. Nerode

Abstract

Vardi, M.Y., Verification of concurrent programs: the automata-theoretic framework, Annals of Pure and Applied Logic 51 (1991) 79–98.

We present an automata-theoretic framework to the verification of concurrent and nondeterministic programs. The basic idea is that to verify that a program P is correct one writes a program A that receives the computation of P as input and diverges only on incorrect computations of P. Now P is correct if and only if a program P_A, obtained by combining P and A, terminates. We formalize this idea in a framework of ω-automata with a recursive set of states. This unifies previous works on verification of fair termination and verification of temporal properties.

1. Introduction

In this paper we present an automata-theoretic framework that unifies several trends in the area of concurrent program verification. The trends are *temporal logic, model checking, automata theory* and *fair termination*. Let us start with a survey of these trends.

In 1977, Pnueli suggested the use of *temporal logic* in the verification of concurrent programs [31]. The basic motivation is that in the verification of concurrent programs it is easier to reason about computation sequences than about input-output relations. Temporal logic is a modal logic that enables one to describe how a situation changes over time [34]. Hence it is appropriate for reasoning about concurrent programs.

Since 1977, there has been significant progress in the development of techniques and methodologies for proving temporal properties of concurrent programs [16, 21–24, 27, 32]. The developed methods reduce program correctness to truth of sentences in first-order temporal logic. Thus, these methods require

* A preliminary version of this paper appeared in Proc. 2nd IEEE Symp. on Logic in Computer Science, Ithaca, 1987, pp. 167–176.

** Part of this research was done while the author was visiting the Weizmann Institute of Science.

0168-0072/91/$03.50 © 1991 — Elsevier Science Publishers B.V. (North-Holland)

temporal reasoning, and do not provide a reduction of a proof of a temporal property into a sequence of proofs of *nontemporal verification conditions* in the underlying assertion languages. This should be contrasted with proof systems for sequential programs, one of whose main features is precisely such a reduction (cf. [3, 4]). We call this *proof by reduction*. For some isolated classes of properties such reductions have been found [23, 24], but the general case remained open.

A concurrent development is the development of proof techniques for *finite-state* programs. It was already shown by Pnueli [31] that verifying arbitrary temporal properties of finite-state programs is decidable. More efficient algorithms were developed in [11, 20, 33] (lower bounds were proven in [39]). These algorithms are called *model-checking algorithms*, since they check whether the program is a model of its specification.

The relevance of automata theory to the verification of concurrent programs was recognized by Park [28, 30] and Nivat [26]. The trend of 'getting away' from temporal logic was started by Wolper [46], who argued the temporal logic lacks expressive power, and introduced *extended temporal logic* (ETL), which uses finite-state ω-automata as a specification language. This trend was continued by Vardi and Wolper [42, 43], who described an automata-theoretic approach to model checking. They use the fact that one can effectively translate a temporal specification into an equivalent specification by a finite-state automaton over infinite execution sequences [44]. Vardi and Wolper have shown how by combining the finite-state program and the finite-state specification, the verification problem can be reduced to an automata-theoretic problem. Essentially, their method is to 'get away' from temporal logic, since it seems difficult to directly verify properties specified in temporal logic. Alpern and Schneider [1, 2] and Manna and Pneuli [25] continued this trend. They describe a proof by reduction method for properties (of arbitrary programs) specified by finite-state automata.

At the same time, a lot of attention has been given to the development of methods for proving *fair* termination of nondeterministic programs ([14] is a good survey of the area). (Since nondeterministic programs are often used to model concurrent programs, this research is also applicable to the latter ones.) A program is fairly terminating if it admits no infinite computation, provided the scheduling of nondeterministic choices is 'fair'. There are many different notions of fairness, and each one requires a different proof rule for termination.

One approach to fair termination is the method of *explicit schedulers* [5, 7, 8, 29]. The main idea of this approach is to reduce fair termination to ordinary termination by augmenting the program with random assignments. A unifying treatment of this method was given by Harel in [17] and pursued in [12]. Harel introduced an infinitary language L in which one can express almost all notions of fairness that have appeared in the literature. He then showed how fair termination can be reduced to termination. More precisely, give a program P and a fairness assertion φ, a program P' is constructed such that P admits no infinite computation that obeys φ if and only if P' admits no infinite computation [12, 17]. Since we know how to prove termination by reduction to an underlying

assertion language [6], Harel's method provide a reduction technique for fair termination.

Another approach to fair termination is the method of *helpful directions* [15, 19]. The main idea of this approach is to define some ranking of program states by means of elements of some well-founded sets. This ranking has to decrease along a computation according to rules that depends on the notion of fairness under consideration. A uniform treatment of this method was given by Rinat et al. [35]. They introduced a proof rule for arbitrary fairness properties expressed in a fragment L^- of Harel's L.

The automata-theoretic framework that we present here unifies all the trends mentioned above. As in [42, 43], the basic idea is to combine the specification with the program. We still deal with specification by automata, but not necessarily finite-state automata. This requires a development of a theory of *recursive ω-automata.* Just as temporal logic formulas can be expressed by finite-state automata, formulas in *recursive temporal logic,* which is an infinitary version of temporal logic, can be expressed by recursive automata. It turns out that all methods for proving fair termination have at their foundations reductions between automata with different acceptance conditions. Thus, the theory gives a simple method of proving by reduction any property specified by recursive automata, and in particular any temporal property.

The outline of the paper is as follows. In Section 2 we develop a theory of recursive automata on infinite words. We describe three types of acceptance conditions: called *Wolper* acceptance, *Büchi* acceptance, and *Streett* acceptance. These acceptance conditions generalize in the natural way corresponding conditions for finite-state automata. We then prove that these conditions all have the same power. Our proofs use and have to deal with the fact that the automata have infinitely many states. We then show that recursive automata can capture properties expressed in recursive temporal logic. In Section 3 we show how our automata-theoretic results can be used to transform infinite trees in the spirit of [17]. Our results extend Harel's results, with what we believe are conceptually simpler proofs. The interest in infinite trees stems from the correspondence between nondeterministic programs and their computation trees. We show in Section 4 how our results on tree transformation can be used to verify temporal properties of programs. We develop proof methods that follow the two major approaches to fair termination: the method of explicit schedulers and the method of helpful directions.

2. Recursive automata on infinite words

2.1. Basic definitions

An *infinite word* w is a function $w: \omega \to \omega$. (One can view w as an infinite word over the alphabet $\{i \mid \exists j \text{ such that } w(j) = i\}$.) Given $i \geqslant 0$, we denote by $\bar{w}(i)$ the sequence $w(0) \ldots w(i)$. A *language* is a set of words, i.e. a subset of ω^ω. A language L is Σ_1^1 if there is an arithmetical relation $R \subseteq \omega^\omega \times \omega^\omega$ such that

$L = \{w \mid \exists u\, R(w, u)\}$ [36]. *Kleene's Normal Form Theorem* states that a language L is Σ_1^1 if and only if there is a recursive relation $R \subseteq \omega^* \times \omega^*$ such that $L = \{w \mid \exists u\, \forall n\, R(\bar{w}(n), \bar{u}(n))\}$ [36].

A *table* T is a tuple (S, S^0, α), where S is a (possibly infinite) set of *states*, $S^0 \subseteq S$ is the set of *starting* states, and $\alpha \subseteq S \times \omega \times S$ is the *transition relation*. T is said to be *recursive* in case S, S^0 and α are recursive. A *run* r of T *on* the word w is a sequence $r: \omega \to S$ such that $r(0) \in S^0$ and $(r(i), w(i), r(i+1)) \in \alpha$ for all $i \geq 0$.

Automata are tables with *acceptance conditions*. A *Wolper automaton* is just a table $T = (S, S^0, \alpha)$. It *accepts* a word w if it has a run on w. A *Büchi automaton* A is a pair (T, F), where $F \subseteq S$. A accepts a word w if there is a run r of T on w such that for infinitely many i's we have $r(i) \in F$. A is recursive if T and F are recursive. A *Streett automaton* A is a tuple (T, I, L, U), where I is some index set, $L \subseteq I \times S$ and $U \subseteq I \times S$. Intuitively, L and U can be viewed as collections of sets indexed by I, i.e., for every $i \in I$, there are sets of states $L_i = \{s \mid (i, s) \in L\}$ and $U_i = \{s \mid (i, s) \in U\}$. In fact, we can also denote A by $(T, \{(L_i, U_i) \mid i \in I\})$. A is recursive if T is recursive and also I, L and U are recursive. A accepts a word w if T has a run r on w such that for every $i \in I$, if there are infinitely many j's such that $r(j) \in L_i$, then there are infinitely many j's such that $r(j) \in U_i$. The language accepted by an automaton A is denoted $L_\omega(A)$. For recursive Streett automata, we can assume without loss of generality that $I < \omega + 1$, i.e., either $I = \omega$ or I is a natural number. Two automata A and A' are *equivalent* when $L_\omega(A) = L_\omega(A')$.

The above definitions generalize the definitions given in [44] of finite-state Wolper automata[1], in [9] of finite-state Büchi automata, and in [41] of finite-state Streett automata.

Clearly, a Wolper automaton $T = (S, S^0, \alpha)$ is equivalent to the Büchi automaton (T, S). Also, a Büchi automaton (T, F) is equivalent to the Streett automaton $(T, \{(S, F)\})$. It is well known that finite-state Büchi automata and finite-state Streett automata have the same expressive power [10], which is stronger than the expressive power of finite-state Wolper automata [44]. The proofs of both facts, the inequivalence of finite-state Wolper and Büchi automata and the equivalence of finite-state Büchi and Streets automata, depend on the finiteness of the state set. In what follows we show that if we allow infinitely many states, then the three classes of automata have the same expressive power. That is, Büchi and Street automata are equivalent even if we allow infinitely many states, and Wolper and Büchi automata become equivalent once we allow infinitely many states.

2.2. *Automata-theoretic reductions*

We first show that every Σ_1^1 language is definable by a Wolper automaton. (This theorem was observed by Plotkin. It is closely related to the results in [45].)

[1] Wolper automata are called *looping* automata in [44].

Theorem 2.1. *Let L be a Σ_1^1 language. Then there is a Wolper automaton T such that $L = L_\omega(A)$.*

Proof. We use Kleene's Normal Form Theorem for Σ_1^1: a language L is Σ_1^1 if there is a recursive relation $R \subseteq \omega^* \times \omega^*$ such that $L = \{w \mid \exists u \, \forall n \, R(\bar{w}(n), \bar{u}(n))\}$. Note that in particular we must have $R(\lambda, \lambda)$ (λ denotes the null sequence).

Let $T = (R, R^\lambda, \alpha)$, where $R^\lambda = \{\langle \lambda, \lambda \rangle\}$ and α is defined as follows: for $\langle \sigma_1, \sigma_2 \rangle \in R$, $\langle \tau_1, \tau_2 \rangle \in R$ and $n \geq 0$, we have that $(\langle \sigma_1, \sigma_2 \rangle, n, \langle \tau_1, \tau_2 \rangle) \in \alpha$ iff $\tau_1 = \sigma_1 n$ and there exists some $m \geq 0$ such that $\tau_2 = \sigma_2 m$. The reader can check that $L = L_\omega(T)$. □

We can now show that, unlike the case for finite-state automata, Büchi and Streett automata are not more expressive than Wolper automata. By Theorem 2.1, all we have to show is that Büchi and Streett automata define Σ_1^1 languages. While this yields in principle effective constructions, extracting these constructions from the proofs is not straightforward. Thus, for many of the following theorems we give two proofs: the first is a one-liner using Theorem 2.1 and the second uses an explicit construction.

Theorem 2.2. *There is an effective transformation that maps every recursive Büchi automaton to an equivalent recursive Wolper automaton.*

Proof. Let $A = (S, S^0, \alpha, F)$ be a recursive Büchi automaton. Then a word w is in $L_\omega(A)$ if there exists a run $r: \omega \to S$ such that $r(0) \in S^0$, $(r(i), w(i), r(i+1)) \in \alpha$ for all $i \geq 0$, and for all $n \geq 0$ there exists some $m > n$ such that $r(m) \in F$. Clearly, $L_\omega(A)$ is Σ_1^1, and the proof of Theorem 2.1 gives us an effective transformation of A to an equivalent recursive Wolper automaton.

We now describe a simpler transformation that does not require using Theorem 2.1. Let $B = (T, T^0, \beta)$ be defined as follows. The state set T is $S \times \omega$. The starting state set T^0 is $S^0 \times \omega$. Finally, the transition relation satisfies $((p, i), a, (q, j)) \in \beta$ if and only if $(p, a, q) \in \alpha$ and either $j = i - 1$ or $q \in F$. Clearly, if A is recursive, then so is B, and so is the transformation from A to B. Also, it can be shown that $L_\omega(A) = L_\omega(B)$. □

Notice that the above proof uses in a strong way the fact that B has infinitely many states. The theorem does not hold if we restrict ourselves to finite-state automata.

We now show that Büchi automata and Streett automata have the same expressive power even for infinite state set. We first develop some intuition by considering a special case.

Remark 2.3. Let $A = (S, S^0, \alpha, I, L, U\}$ be a Streett automaton where $I < \omega$. We describe the construction of an equivalent Büchi automaton in two steps.

Let $B = (T, T_0, \beta)$ be a table defined as follows. The state set T is $S \times 3^I$. The starting state set is $T^0 = S^0 \times 3^I$. The transition relation β satisfies: $((p, \boldsymbol{x}), a, (q, \boldsymbol{y})) \in \beta$ if and only the following holds for all $i < I$:

- $(p, a, q) \in \alpha$,
- if $x_i = 2$, then $y_i = 2$,
- if $x_i = 1$, then $y_i \leqslant 1$, and
- if $x_i = 0$, then $y_i = 0$ and $q \notin L_i$.

Intuitively, the second component of a state carries a 'prediction' about how many times some L_i will be encountered. A '2' in the ith place denotes a prediction that L_i will be encountered infinitely many times. A '1' in the ith place denotes a prediction that L_i will be encountered only finitely many times. A '0' in the ith place denotes a prediction that L_i will not be encountered any more.

Consider now a run $(p^0, \boldsymbol{x}^0), (p^1, \boldsymbol{x}^1), \ldots$ of B on a word w. The run $p^0, p^1, \ldots$ of A on w is accepting, if for all $j < I$, either $x_j^k = 2$ for all $k \geqslant 0$ and there are infinitely many i's such that $p^i \in U_j$, or the sequence $x_j^0, x_j^1, x_j^2, \ldots$ contains a 0.

Let $C = (W, T^0 \times \{0\}, \gamma)$ be a table defined as follows. The state set W is $T \times (I + 1)$. The transition relation γ satisfies the following condition: $((p, \boldsymbol{x}, i), a, (q, \boldsymbol{y}, k)) \in \gamma$ if

- $((p, \boldsymbol{x}), a, (q, \boldsymbol{y})) \in \beta$,
- if $0 \leqslant i < I$, then either $y_i > 0$ and $q \notin U_i$, in which case $k = i$, or otherwise $y_i = 0$ or $q \in U_i$, in which case $k = i + 1$, and
- if $i = I$, then $k = 0$.

Intuitively, the third component of a state verifies that all predictions are satisfied.

Consider now a run $(p^0, \boldsymbol{x}^0, 0), \ldots (p^i, \boldsymbol{x}^i, k^i), \ldots$ of B on a word w. The run $p^0, p^1, \ldots$ of A on w is accepting, if for infinitely many i's we have $k^i = I$. Thus, A is equivalent to the Büchi automaton (C, F), where $F = T \times \{I\}$.

Note that the above construction preserves finiteness, i.e., if we start with a finite-state Streett automaton, then we get a finite-state Büchi automaton. Note also that the construction fails if $I = \omega$, since 3^I is not countable. We now deal with the general case.

Theorem 2.4. *There is an effective transformation that maps every recursive Streett automaton to an equivalent recursive Büchi automaton.*

Proof. Let $A = (S, S^0, \alpha, I, L, U\}$ be a Streett automaton. Again, it is easy to see that $L_\omega(A)$ is Σ_1^1, so the proof of Theorem 2.1 yields the desired transformation.

We give here a direct construction that has the advantage that it yields a finite-state Büchi automaton when given a finite-state Streett automaton. We describe the construction of an equivalent Büchi automaton in two steps.

Let $B = (T, T_0, \beta)$ be a table defined as follows. The state set T is $S \times 3^{<I}$.[2] The starting state set is $T^0 = S^0 \times \{\lambda\}$, where λ is the null sequence. The transition relation β satisfies: $((p, \boldsymbol{x}), a, (q, \boldsymbol{y})) \in \beta$ if and only if the following holds for all $i < I$:

- $(p, a, q) \in \alpha$,
- If $|\boldsymbol{x}| < I$, then $|\boldsymbol{y}| = |\boldsymbol{x}| + 1$, otherwise $|\boldsymbol{y}| = |\boldsymbol{x}|$,
- if $x_i = 2$, then $y_i = 2$,
- if $x_i = 1$, then $y_i \leq 1$, and
- if $x_i = 0$, then $y_i = 0$ and $q \notin L_i$.

The intuition behind the construction is similar to the intuition in Remark 2.3. Notice how 3^I has been replaced here by $3^{<I}$.

Consider now a run $(p^0, \boldsymbol{x}^0), (p^1, \boldsymbol{x}^1), \ldots$ of B on a word w. The run $p^0, p^1, \ldots$ of A on w is accepting, if for all $j < I$, either $x_j^k = 2$ for all $k \geq j$ and there are infinitely many i's such that $p^i \in U_j$, or the sequence $x_j^j, x_j^{j+1}, x_j^{j+2}, \ldots$ contains a 0.

Let $C = (W, T^0 \times \{(0, 0)\}, \gamma)$ be a table defined as follows. The state set W is $T \times I^2$. The transition relation γ satisfies the following condition: $((p, \boldsymbol{x}, i, j), a, (q, \boldsymbol{y}, k, l)) \in \gamma$ if

- $((p, \boldsymbol{x}), a, (q, \boldsymbol{y})) \in \beta$,
- if $i < j$, then $l = j$,
- if $0 \leq i < j$, then either $y_i > 0$ and $q \notin U_i$, in which case $k = i$, or otherwise $y_i = 0$ or $q \in U_i$, in which case $k = i + 1$, and
- if $i = j$, then $k = 0$ and either $j = I$ and $l = j$ or $j < I$ and $l = j + 1$.

Intuitively, the third component of a state verifies that all predictions are satisfied.

Consider now a run $(p^0, \boldsymbol{x}^0, 0, 0), \ldots (p^i, \boldsymbol{x}^i, k^i, l^i), \ldots$ of B on a word w. The run $p^0, p^1, \ldots$ of A on w is accepting, if for infinitely many i's we have $k^i = l^i$. Thus, A is equivalent to the Büchi automaton (C, F), where $F = T \times \{(i, i) \mid i \in I\}$. □

We note that C is infinite only because A is infinite. If A is finite, then C is also finite. This should be contrasted with the earlier transformation from Büchi automata to Wolper automata.

It is known that the classes of finite-state Büchi and Streett automata are closed under finite unions, finite intersections and complementation [10]. We now state some closure results for recursive automata.

Theorem 2.5. *There is an effective transformation that, given a recursive sequence $A_0, A_1, \ldots$ of recursive Streett automata, gives a recursive Streett automaton A such that $L_\omega(A) = \bigcup_{i \in \omega} L_\omega(A_i)$. Similarly, there is an effective transformation that, given a recursive sequence $A_0, A_1, \ldots$ of recursive Streett automata, gives a recursive Streett automaton A such that $L_\omega(A) = \bigcap_{i \in \omega} L_\omega(A_i)$.*

[2] For a set X and an ordinal ν, we use $X^{<\nu}$ to denote $\bigcup_{\mu < \nu} X^\mu$. In particular $X^{<\omega} = X^*$.

Proof. As in the previous theorems, the claim of the above theorem follows by Theorem 2.1. We describe here direct transformations that have the feature that when given a finite sequence of finite-state automata they yield finite-state automata.

Let $\{A_j, j \in J\}$ be a recursive sequence of Streett automata, i.e., J is recursive and $\{(A_j, j) \mid j \in J\}$ is recursive. We can assume without loss of generality that $J < \omega + 1$, i.e., either $J = \omega$ or J is a natural number. Let $A_j = (S_j, S_j^0, \alpha_j, I_j, L_j, U_j)$.

We first prove closure under union. Let $A = (S, S^0, \alpha, I, L, U)$ be the disjoint union of the A_j's. The state set S is $\{\langle s, j\rangle \mid j \in J, s \in S_j\}$. The starting state set S^0 is $\{\langle s, j\rangle \mid j \in J, s \in S_0^j\}$. The transition relation is $\{(\langle s, j\rangle, k, \langle t, j\rangle)\colon j \in J, (s, k, t) \in \alpha_j$. Finally, we have $I = \{\langle i, j\rangle \mid i \in I_j\}$, $L = \{(\langle i, j\rangle, \langle s, j\rangle) \mid (i, s) \in L_j\}$ and $U = \{(\langle i, j\rangle, \langle s, j\rangle) \mid (i, s) \in U_j\}$. It is easy to see that $L_\omega(A) = \bigcup_{j \in J} L_\omega(A_j)$. Note that A is recursive and if J is finite and all A_j's are finite, then A is also finite.

We now prove closure under intersection. We first have to extend the domain of the transition relations. If $\alpha \subseteq S \times \omega \times S$ is a transition relation, then $\alpha^* \subseteq S \times \omega^* \times S$ is an *extended* transition relation defined inductively:

- $(s, \lambda, s) \in \alpha^*$ and
- $(s, xi, t) \in \alpha^*$ if for some $u \in S$ we have that $(s, x, u) \in \alpha^*$ and $(u, i, t) \in \alpha$.

The automaton $A = (S, S^0, \alpha, I, L, U)$ is obtained by letting all the A_j's run 'almost' concurrently. A naive approach is to have A be a cross product of the A_i's, but then we cannot have A be recursive. Instead we 'start' the A_i's one after another. At every point we have only finitely many A_i's running, but every A_i is eventually started. A formal description follows.

The state set S is $\omega^{<J} \times \bigcup_{0 \leq k < J} \prod_{0 \leq j \leq k} S_j$, i.e., a pair consisting of a sequence of numbers and a sequence of states. The first element in the pair is intended to be the prefix of the word that was read so far by A and the second element of the pair is a sequence of states of the A_i's that have already been started. The starting state set S^0 is $\{\lambda\} \times S_0^0$. The transition relation α is the union of

$$\{(\langle x, s_0, \ldots, s_{k-1}\rangle, i, \langle xi, t_0, \ldots, t_k\rangle) \mid k < J, (s_i, i, t_i) \in \alpha_i \text{ for } 0 \leq i < k, \text{ and } (t_0, xi, t_k) \in \alpha_k^* \text{ for some } t_0 \in S_k^0\}$$

and

$$\{(\langle x, s_0, \ldots, s_{J-1}\rangle, i, \langle x, t_0, \ldots, t_{J-1}\rangle) \mid J < \omega \text{ and } (s_i, i, t_i) \in \alpha_i \text{ for } 0 \leq i < I\}.$$

Thus, the transition relations changes the state of all the A_i's that were already started, and starts a new A_i if not all of them have been started. If not all of the A_i's have been started, then we also remember the prefix of the input read so far.

It remains to define the acceptance condition. Let $S^{\geq j}$ denote $\bigcup_{j \leq k < J} \prod_{0 \leq i \leq k} S_i$, i.e., $S^{\geq j}$ is the subset of S that consists of tuples whose length is at least $j + 1$. We now define I to be the set $\{\langle i, j\rangle \mid j \in J, i \in I_j\}$, we define L to be the set $\{(\langle i, j\rangle, \boldsymbol{s}) \mid \langle i, j\rangle \in I, \boldsymbol{s} \in S^{\geq j}, (i, \boldsymbol{s}_j) \in L_j\}$, and we define U to be the set $\{(\langle i, j\rangle, \boldsymbol{s}) \mid \langle i, j\rangle \in I, \boldsymbol{s} \in S^{\geq j}, (i, \boldsymbol{s}_j) \in U_j\}$.

We leave it to the reader to verify that $L_\omega(A) = \bigcap_{j \in J} L_\omega(A_j)$. Note that A is recursive and if J is finite and all A_j's are finite, then A is also finite. □

One should note that not all results from finite-state automata theory carry over to recursive automata. For example, it follows immediately from Theorem 2.1 that recursive automata are not closed under complement, since it is known that Σ_1^1 is not closed under complement [36]. Also recursive automata are not determinizable, since determinizability would imply closure under complement.

2.3. Recursive temporal logic

Wolper introduced an infinitary version of temporal logic, which he called IPTL [47]. We define here *recursive infinitary temporal logic* RITL, which is the effective fragment of IPTL.

Let $\theta_0, \theta_1, \ldots$ be a recursive sequence of predicates on ω. (That is, $\theta_i \in 2^\omega$ and the relation $\{(i, j) \mid j \in \theta_i\}$ is recursive.) The θ_i's are the atomic formulas of RITL. The closure of the atomic formulas under negations, under the $\bigcirc$ connective, and finite or recursively infinite conjunctions constitutes RITL. Let w be a word, then w^i is a word defined by $w^i(j) = w(i + j)$. Satisfaction of formulas by a word w is defined as follows:

- $w \models \theta_i$ if $w(0) \in \theta_i$,
- $w \models \neg\varphi$ if it is not the case that $w \models \varphi$,
- $w \models \bigcirc\varphi$ if $w^1 \models \varphi$,
- $w \models \bigwedge_{i \in I} \varphi_i$ if $w \models \varphi_i$ for all $i \in I$.

The language RITL is a very powerful language. Note that all the standard temporal connectives can be expressed in RITL. For example, the temporal logic formula $\varphi U \psi$ (φ 'until' ψ) can be expressed as $\bigvee_{i \geq 0} (\bigcirc^i \psi \wedge \bigwedge_{0 \leq j < i} \bigcirc^j \varphi)$, where $\bigcirc^0 \theta$ is θ and $\bigcirc^k \theta$ is $\bigcirc \bigcirc^{k-1} \theta$ for $k > 1$. Also, RITL can express very rich notions of fairness. For example, RITL can express *equifairness* [14] without auxiliary variables. Harel's infinitary assertion language L [17], which is constructed by finite and recursively infinite conjunctions and disjunctions from the atomic formulas $\exists\theta_i$ ('θ_i is true at some point'), $\forall\theta_i$ ('θ_i is true at all points'), $\exists^\infty\theta_i$ ('θ_i is true at infinitely many points'), and $\forall^\infty\theta_i$ ('θ_i is true at all but finitely many points'), is a fragment of recursive infinitary temporal logic. Essentially, RITL is obtained from L by augmenting it with the ability to talk about the 'present moment' and the 'next moment'.

Given a formula φ of RITL, let $L_\omega(\varphi)$ be the set of words satisfied by φ.

Theorem 2.6. *There is an effective transformation that maps every formula φ of* RITL *to a recursive Wolper automaton A such that $L_\omega(\varphi) = L_\omega(A)$.*

Proof. It is not hard to see that $L_\omega(\varphi)$ is Σ_1^1. Thus, the claim follows by Theorem 2.1[3]. We now show a direct transformation that does not use Theorem 2.1.

[3] In fact, the claim also holds for *arithmetical infinitary temporal logic*, which is the arithmetical analogue of RITL.

We first augment RITL with infinite disjunctions with the obvious semantics. The augmented logic has the feature that every formula is equivalent to a formula where negations are applied only to atomic formulas. (This follows from the equivalences $\neg(\bigwedge_{i\in I}\varphi_i)\equiv\bigvee_{i\in I}\varphi_i$ and $\neg\bigcirc\varphi\equiv\bigcirc\neg\varphi$.) We can now prove the claim by induction on the structure of the formula.

For the base case, assume that φ is an atomic formula θ_i (respectively, a negation of an atomic formula $\neg\theta_i$). Then $A_\varphi=(S_\varphi, S^0_\varphi, \alpha_\varphi)$, where $S_\varphi=\{0,1\}$, $S^0_\varphi=\{0\}$, and $\alpha_\varphi=\{(0,j,1)\mid j\in\theta_i\}\cup\{(1,j,1)\mid j\geqslant 0\}$ (respectively, $\alpha_\varphi=\{(0,j,1)\mid j\notin\theta_i\}\cup\{(1,j,1)\mid j\geqslant 0\}$).

Suppose now that φ is $\bigcirc\psi$ and we have already constructed $A_\psi=(S_\psi, S^0_\psi, \alpha_\psi)$. Then $A_\varphi=(S_\varphi, S^0_\varphi, \alpha_\psi)$, where $S_\varphi=\{0\}\cup\{k+1\mid k\in S_\psi\}$, $S^0_\varphi=\{0\}$ and $\alpha_\varphi=\{(0,j,k)\mid k-1\in S^0_\psi\cup\{(k,j,l)\mid(k-1,j,l-1)\in\alpha_\psi\}$.

Finally, if φ is $\bigwedge_{i\in I}\psi_i$ or $\bigvee_{i\in I}\psi_i$, then we can use Theorem 2.5 to construct A_φ from the A_{ψ_i}'s. □

In [44] it is shown that every temporal logic formula can be effectively translated to an equivalent finite-state automaton. Theorem 2.6 is the natural generalization to RITL. Note that the theorem holds in spite of the fact that RITL is closed under negation and recursive automata are not. The reason is that negation can be pushed down to atomic formulas.

3. Infinite trees

We associate computation trees with nondeterministic programs in the natural way. Conditions about the correctness of the program can then be expressed as conditions on the paths of the computation tree. The main technical result in [17] is a transformation of trees with complicated correctness conditions to trees with simple correctness conditions. In this section we apply the results of Section 2 to derive certain transformation on trees in the spirit of [17]. In the next section we show how to apply these transformations to program verification.

A *node* is an element of ω^*. A *tree* is a set of nodes closed under the prefix operation. The *root* of the tree is λ, and a *path* is a maximal increasing sequence of successive nodes (by the prefix ordering) starting at λ. Thus, a path is a sequence of elements in ω^*. A tree is *well founded* if all its paths are finite. We adopt some standard encoding e of ω^*, so we can view an infinite path $x_0, x_1, \ldots$ as the infinite word $e(x_0)e(x_1)\cdots$. A tree is *recursive* if its characteristic function is recursive.

A *recurrence-free* tree is a pair (τ, M), consisting of a tree τ and a set $M\subseteq\omega$ such that no path in τ has an infinite intersection with M. (A path that has an infinite intersection with M is called *recurrent*.) It is recursive if both τ and M are recursive. An *avoiding* tree is a pair (τ, A), consisting of a tree τ and an automaton A such that no infinite path in τ is accepted by A. (Paths accepted by

A are called *A-abiding.*) It is recursive if both τ and A are recursive. Clearly, recurrence-free trees can be seen as subclass of the avoiding trees. Note that by Theorem 2.6, the notion of avoiding trees is a generalization of Harel's notion of φ-avoiding trees for assertions φ in L.

Intuitively, a well-founded tree is the computation tree of a terminating program. An avoiding tree (τ, A) is the computation tree of a fairly terminating program, where A accepts precisely the fair computations. Our goal is to establish a transformation between recursive avoiding trees and recursive well-founded trees. Now, it is known that the set of (notations for) recursive well-founded trees is Π_1^1-complete [36]. Also, it follows from results in [18] that the sets of recursive recurrence-free trees and recursive avoiding trees are Π_1^1-complete. Thus, by definition, the sets of recursive well-founded trees, the set of recursive recurrence-free trees, and the set of recursive avoiding trees are pairwise recursively isomorphic. Our interest, however, is in *simple* transformations from avoiding trees to recurrence-free trees and to well-founded trees that preserve structure as much as possible. In particular, we would like the transformations to preserve the structure of paths (which represent computations).

We first describe a transformation from avoiding trees to recurrence-free trees.

Theorem 3.1. *There is a recursive one-to-one transformation from the set of avoiding trees to the set of recurrence-free trees.*

Proof. Let τ be a tree, and let A be a recursive automaton. By Theorem 2.4, we can assume that A is a Büchi automaton (S, S_0, α, F). We define a tree τ_A as a subset of $\omega^* \times S^*$. (Strictly speaking, a tree has to be a subset of ω^*, but we will ignore this technicality here.) The root of the tree is $\langle \lambda, \lambda \rangle$. A pair (i, pq) is a child of $\langle \lambda, \lambda \rangle$ if $i \in \tau$, $(p, e(i), q) \in \alpha$, and $p \in S_0$, and (ui, ypq), where $u \in \omega^+$ and $y \in S^*$, is a child of (u, yp) if $ui \in \tau$ and $(p, e(ui), q) \in \alpha$.

The sequence $(\lambda, \lambda), (u_1, p_0p_1), (u_2, p_0p_1p_2), \ldots$ is a path in τ_A if and only if $\lambda, u_1, u_2, \ldots$ is a path in τ, $p_0 \in S_0$, and the sequence $p_0, p_1, p_2, \ldots$ is a run of A on $e(u_1), e(u_2), \ldots$. Thus, $u_1, u_2, \ldots$ is A-abiding iff $(\lambda, \lambda), (u_1, p_0p_1), (u_2, p_0p_1p_2), \ldots$ is a recurrent with respect to $\omega^* \times S^*F$. If (τ, a) is avoiding, then $(\tau_A, \omega^* \times S^*F)$ is recurrence-free. □

We now describe the transformation from avoiding trees to well-founded trees.

Theorem 3.2. *There is a recursive one-to-one transformation from the set avoiding trees to the set of well-founded trees.*

Proof. Let τ be a tree, and let A be a recursive automaton. By Theorem 2.2, we can assume that A is a Wolper automaton (S, S_0, α). The construction in the proof of the previous theorem yields a well-founded tree. □

Theorems 3.1 and 3.2 extends Harel's result [17], and with much simpler proofs (cf. [17, proof of Theorem 8.1]). So where is the catch? The direct proof in [17] essentially combines the automata-theoretic and the tree-theoretic transformations. Our approach is to separate the automata-theoretic part from the tree-theoretic part. Once we have the automata-theoretic results, the tree-theoretic results are straightforward.

4. Program verification

4.1. The basic approach

Rather than restrict ourselves to a particular programming language, we use here an abstract model for nondeterministic programs (we model concurrency by nondeterminism). A *program* P is a triple (W, I, R), where W is a set of *program states*, $I \subseteq W$ is a set of *initial states*, and $R \subseteq W^2$ is a binary *transition relation* on W. A computation is a sequence σ in W^ω such that $\sigma(0) \in I$ and $(\sigma(i), \sigma(i+1)) \in R$ for all $i \geq 0$. (For simplicity we assume that the program has only infinite computations. A terminating computation is assumed to stay forever in its last state.) Given that programs are supposed to be effective, and assuming that the programs run over an arithmetical domain, we require that W, R and I are recursive sets. The reader should note the similarity of programs and Wolper automata.

We assume some underlying *assertion language*, say a first-order logic, in which one can express assertions about program states. The assertion language gives the building blocks to the *fairness language* and the *specification language*. The fairness language is used to specify what computations are considered to be 'fair', i.e., when is the scheduling of nondeterministic choices not too pathological (we assume that the assertion language can express statements about the scheduling). Thus, only computations that satisfy the fairness condition need be considered when the program is verified. The specification language is used to express the correctness required of the computation, in other words, this is what the user demands of the computation.

Given a fairness condition Φ and a correctness condition Ψ, the program P is *correct* with respect to (Φ, Ψ) if every computation of P that satisfies Φ also satisfies Ψ. The crux of our approach is to prove that the program is not *incorrect*, i.e., there is no computation of P that satisfies $\Phi \wedge \neg\Psi$. Or, to put it in the automata-theoretic framework, if τ_P is the computation tree of the program (defined in the obvious way) and A is an automaton that accepts precisely the words satisfying $\Phi \wedge \neg\Psi$, then we have to show that (τ_P, A) is avoiding.

We have to decide now in what languages are Φ and Ψ specified. If we use RITL as both the fairness language and specification language, then, since RITL is closed under negation, we can apply Theorem 2.6 and voilà. Also, if Ψ is a

finite-state automaton, then it can be complemented [40]. If, however, we want to directly use the power of recursive automata in the specification, then we have to directly specify incorrectness, since we cannot complement recursive automata. In fact, if Φ is given by a recursive automaton, then the complexity of the verification problem is Π_2^1, which means that our verification techniques are not applicable [38]. Indeed, Manna and Pnueli's decision to use ∀-automata [25], which are essentially Büchi automata that specify incorrectness, was influenced by an early exposition of the ideas in this paper. Thus, we assume that we already have a recursive automaton $A_{\Phi,\Psi}$ that expresses $\Phi \wedge \neg \Psi$.

In what follows we describe two approaches to verification, in the spirit of the method of explicit schedulers and the method of helpful directions.

4.2. Explicit schedulers

The idea is to transform P to a program $P_{\Phi,\Psi}$ such that P is correct with respect to (Φ, Ψ) iff $P_{\Phi,\Psi}$ *fairly terminates*. A program $P = (W, I, R)$ fairly terminates with respect to a fairness condition Φ if it has no infinite computation satisfying Φ. In particular, if $U \subseteq W$, then P fairly terminates with respect to U if it has no infinite computations with infinitely many states in U. The transformation is essentially the transformation in the proof of Theorem 3.1. By Theorem 2.4, we can assume that $A_{\Phi,\Psi} = (S, S^0, \alpha, F)$ is a Büchi automaton. Now, $P_{\Phi,\Psi}$ is obtained by combining P with $A_{\Phi,\Psi}$. More precisely, $P_{\Phi,\Psi} = (W \times S, I \times S^0, R_\alpha)$, where $((u, p), (v, q)) \in R_\alpha$ iff $(u, v) \in R$ and $(p, u, q) \in \alpha$. The nondeterministic choices in P are now directed by $A_{\Phi,\Psi}$. Thus, $A_{\Phi,\Psi}$ can be viewed as a scheduler for P.

Theorem 4.1. *P is correct with respect to (Φ, Ψ) iff $P_{\Phi,\Psi}$ fairly terminates with respect to $(W \times F)$.*

Proof. Suppose that P is not correct with respect to (Φ, Ψ). That is, there is a computation σ of P such that σ satisfies the fairness condition Φ but not the correctness assertion Ψ. Thus, $\sigma \in L_\omega(A_{\Phi,\Psi})$, so there is an accepting run r of $A_{\Phi,\Psi}$ on σ. Let ξ be a member of $(W \times S)^\omega$ defined by $\xi(i) = (\sigma(i), r(i))$. It is easy to see that ξ is a computation of $P_{\Phi,\Psi}$ with infinitely many states in $W \times F$, so $P_{\Phi,\Psi}$ does not fairly terminate with respect to $(W \times F)$.

Conversely, suppose that $P_{\Phi,\Psi}$ does not fairly terminate with respect to $(W \times F)$ and $\xi \in (W \times S)^\omega$ is an infinite computation of $P_{\Phi,\Psi}$ with infinitely many states in $W \times F$. Let $\xi(i) = (\sigma_i, r_i)$. Let $\sigma \in W^\omega$ be $\sigma_0, \sigma_1, \ldots$, and let $r \in S^\omega$ be $r_0, r_1, \ldots$. It is easy to see that r is an accepting run of $A_{\Phi,\Psi}$ on a computation σ of P. But that means that σ satisfies the fairness condition Φ but not the correctness condition Ψ, so P is not correct. □

In practice, a program is not an abstract set of states, but an actual syntactical object. Theorem 4.1 gives a method of syntactically transforming the program P

to the program $P_{\Phi,\Psi}$. The combining of P and $A_{\Phi,\Psi}$ is done by adding to P *auxiliary variables* that keep the information about the automaton states. In general, $P_{\Phi,\Psi}$ can be a very complicated program, since $A_{\Phi,\Psi}$ can be a very complicated automaton. But in most cases we expect $A_{\Phi,\Psi}$ to be a finite-state automaton, so the transformation from P to $P_{\Phi,\Psi}$ is not too complicated.

To prove (fair) termination of $P_{\Phi,\Psi}$, we use the proof by reduction technique of [6]. (Note, however, that the proof requires intermediate assertions in fixpoint logic.) In fact, since we can assume, by Theorem 2.2, that $A_{\Phi,\Psi}$ is a Wolper automaton, we could have taken F to be S, in which case fair termination with respect to $W \times F$ is standard termination.

Corollary 4.2. *If $A_{\Phi,\Psi}$ is a Wolper automaton, then P is correct with respect to (Φ, Ψ) iff $P_{\Phi,\Psi}$ terminates.*

In particular, if the correctness condition is termination, then Corollary 4.2 is a reduction of fair termination to termination, generalizing [5, 7, 8, 12, 17, 29].

Theorem 4.1 is of special interest when P and $A_{\Phi,\Psi}$ are finite states, because in that case $P_{\Phi,\Psi}$ is also finite state. Checking fair termination of $P_{\Phi,\Psi}$ can now be done algorithmically. This is the essence of Vardi and Wolper's approach to the verification of finite-state programs [42, 43].

Example 4.3. Let P be the following program written in Dijkstra's guarded command language [13].

$n \leftarrow 0$;
DO
true $\rightarrow n \leftarrow n + 1$;
$\square$**true** $\rightarrow n \leftarrow -n$
OD;

The correctness condition Ψ is that eventually n becomes negative, or formally, $F(n < 0)$, where F is the '*eventually*' connective of temporal logic[4]. The fairness condition Φ is that both choices of the guarded command are taken infinitely often, or formally, $GFat_1 \wedge GFat_2$, where G is the '*always*' connective of temporal logic and at_1 (respectively, at_2) is true when the first (respectively second) choice of the guarded command is taken[5]. Consider the Wolper automaton $A_{\Phi,\Psi} = (S, S^0, \alpha)$, where $S = \{1, 2\} \times \omega$, $S^0 = \{1\} \times \omega$, and $(\langle k, p\rangle, i, \langle l, q\rangle) \in \alpha$ if:

(1) $k = l = 1$, $q = p - 1 \geq 0$, $i \in (n \geq 0)$ and $i \notin at_1$,
(2) $k = 1$, $l = 2$, $q = p - 1 \geq 0$, $i \in (n \geq 0)$ and $i \in at_1$,
(3) $k = l = 2$, $q = p - 1 \geq 0$, $i \in (n \geq 0)$ and $i \notin at_2$,
(4) $k = 2$, $l = 1$, $p \geq 0$, $q \geq 0$, $i \in (n \geq 0)$ and $i \in at_2$.

[4] $w \models F\varphi$ if for some $i \geq 0$ we have $w^i \models \varphi$.
[5] $w \models G\varphi$ if for all $i \geq 0$ we have $w^i \models \varphi$.

It is not hard to see that the automaton $A_{\Phi,\Psi} = (S, S^0, \alpha)$ checks that both at_1 and at_2 are true infinitely often while $n \geq 0$ is always true.

We can now construct $P_{\Phi,\Psi}$ by combining P and $A_{\Phi,\Psi}$:

$n \leftarrow 0; j \leftarrow 1; p \leftarrow ?;$
DO
$n \geq 0 \wedge p > 0 \wedge j = 1 \rightarrow n \leftarrow n + 1; j \leftarrow 2; p \leftarrow p - 1;$
$\square n \geq 0 \wedge p > 0 \wedge j = 2 \rightarrow n \leftarrow n + 1; j \leftarrow 2; p \leftarrow p - 1;$
$\square n \geq 0 \wedge p > 0 \wedge j = 1 \rightarrow n \leftarrow -n; j \leftarrow 1; p \leftarrow p - 1;$
$\square n \geq 0 \wedge p \geq 0 \wedge j = 2 \rightarrow n \leftarrow -n; j \leftarrow 1; p \leftarrow ?$
OD;

By Corollary 4.2, $P_{\Phi,\Psi}$ terminates if and only if P is correct. Since all the guarded statements in $P_{\Phi,\Psi}$ except for the last one decrease the value of p and the last command makes n negative, $P_{\Phi,\Psi}$ terminates, so P is correct.

4.3. Helpful directions[6]

The standard approach to prove termination is to associate rank in a well-founded set with every state of the program and to show that every transition decrease the rank. When dealing with fair termination, we cannot require that every transition decrease the rank; rather we require that transitions cannot increase the rank, and 'helpful' transitions decrease it. To prove fair termination it suffices then to show that in fair computations the 'helpful' transitions are taken infinitely often [15, 19]. To prove correctness we show that in 'bad' computations, i.e., computations that satisfy $\Phi \wedge \neg\Psi$, the 'helpful' transitions are taken infinitely often, which is impossible. We apply first Theorem 2.4, so we can assume that $A_{\Phi,\Psi}$ is a Büchi automaton. We now let $A_{\Phi,\Psi}$ decide what are the 'helpful' directions.

Theorem 4.4. *$P = (W, I, R)$ is correct with respect to (Φ, Ψ), where $A_{\Phi,\Psi} = (S, S^0, \alpha, F)$, iff there exists an ordinal κ and a rank predicate $\rho : 2^{W \times S \times \kappa}$ such that the following holds:*

- *for all $u \in I$ and $p \in S^0$, we have that $\rho(u, p, \kappa)$ holds,*
- *for all $u, v \in W$ and $p, q \in S$, if $\rho(u, p, \mu)$ holds, $(u, v) \in R$ and $(p, u, q) \in \alpha$, then $\rho(v, q, \nu)$ holds for some $\nu \leq \mu$,*
- *for all $u, v \in W$ and $p, q \in S$, if $\rho(u, p, \mu)$ holds, $(u, v) \in R$, $(p, u, q) \in \alpha$ and $p \in F$, then $\rho(v, q, \nu)$ holds for some $\nu < \mu$.*

Proof. By Theorem 4.1 we know that P is correct with respect to (Φ, Ψ) if and only if $P_{\Phi,\Psi} = (W \times S, I \times S^0, R_\alpha)$ fairly terminates with respect to $(W \times F)$. So it suffices to show that the condition in the theorem is necessary and sufficient for fair termination.

[6] For a related automata-theoretic treatment of the helpful-directions approach see [37].

Suppose first that $P_{\Phi,\Psi}$ does not fairly terminate with respect to $(W \times F)$ and $\xi \in (W \times S)^\omega$ is an infinite computation of $P_{\Phi,\Psi}$ with infinitely many states in $W \times F$. Then

- $\rho(\xi_0, \kappa)$ holds,
- if $\rho(\xi_i, \mu)$ holds, then $\rho(\xi_{i+1}, \nu)$ holds for some $\nu \leq \mu$, and
- if $\xi_i \in W \times F$ and $\rho(\xi_i, \mu)$ hold, then $\rho(\xi_{i+1}, \nu)$ holds for some $\nu < \mu$.

But this is impossible, since $\xi_i \in W \times F$ for infinitely many i's.

Suppose now that $P_{\Phi,\Psi}$ fairly terminates with respect to $(W \times F)$. Let $\zeta \in W \times S$. Then we denote by $P^{\zeta}_{\Phi,\Psi}$ the program $(W \times S, \{\zeta\}, R_\alpha)$, i.e., $P^{\zeta}_{\Phi,\Psi}$ is different from $P_{\Phi,\Psi}$ only in its initial state, which is ζ. Consider the set $X \subseteq W \times S$ of states reachable from $I \times S^0$, i.e., $\zeta \in X$ if ζ occurs in a computation of $P_{\Phi,\Psi}$.

We now define a relation $\succ$ on X. We say that $\zeta \succ \eta$ if there is a computation ξ of $P^{\zeta}_{\Phi,\Psi}$ such that $\xi_i = \eta$ for some $i > 0$ and $\xi_j \in W \times F$ for some j, $0 < j \leq i$. That is, $\zeta \succ \eta$ if there is a computation from ζ to η through $W \times F$. It is easy to see that $\succ$ is a partial order, since if there is a infinite chain $\zeta_0 \succ \zeta_1 \succ \cdots$, then there is a computation of $P_{\Phi,\Psi}$ that intersects $W \times F$ infinitely often. Furthermore, $\succ$ is a well-founded partial order. In particular, every subset Y of X has minimal elements, denoted $\min(Y)$.

We now define a (possibly transfinite) sequence of subsets of X. X_0 is just $\min(X)$. Suppose that X_ν has been defined for all $\nu < \mu$ and $\bigcup_{\nu<\mu} X_\nu$ is a proper subset of X. $X_\mu = \min(X - \bigcup_{\nu<\mu} X_\nu)$. Clearly, $\bigcup_{\nu \geq 0} X_\nu = X$. Define the *rank* of an element $\zeta \in X$, denoted $\mathrm{rank}(\zeta)$, to be the ordinal μ such that $\zeta \in X_\mu$. It is easy to see that if for states ζ and η in X we have $(\zeta, \eta) \in R_\alpha$, then $\mathrm{rank}(\zeta) \geq \mathrm{rank}(\eta)$, and if $\zeta \succ \eta$, then $\mathrm{rank}(\zeta) > \mathrm{rank}(\eta)$. We can now define the rank predicate: $\rho(\zeta, \nu)$ holds iff $\zeta \in X$ and $\nu \geq \mathrm{rank}(\zeta)$. Let κ be the length of the sequence $X_0, X_1, \ldots$. We leave it to the reader to verify that κ and ρ, satisfy the conditions of the theorem. □

Note that we have not assigned ranks to programs states, but rather to pairs consisting of a program state and an automaton state as in [1]. Alternatively, one can associate a rank predicate with each state of $A_{\Phi,\Psi}$ in the spirit of [35]. This would be practical if $A_{\Phi,\Psi}$ is finite state.

Theorem 4.4 extends the results in [35]. In that paper, the method of helpful directions was applied to derive a proof rule for fair termination for arbitrary fairness properties expressed in a fragment L^- of Harel's L. L^- is obtained from L by allowing only finite conjunctions and disjunctions. The automata-theoretic approach enables us to deal also with recursively infinite conjunctions and disjunctions.

Theorem 4.4 gives a 'generic' proof rule. By substituting automata that correspond to certain fairness and correctness conditions, one can derive explicit proof rules for such conditions.

Example 4.5 (*impartial termination*). The context here is a concurrent system with n processes. Intuitively, a infinite computation is impartial if every process is scheduled infinitely often along the computation. More formally, we have predicates scheduled$_i$, $1 \leq i \leq n$, and an infinite computation is impartial if it satisfies the temporal formula $\bigwedge_{i=1}^{n} GF$ scheduled$_i$. A program P impartially terminates if it has no infinite impartial computations. We now derive a proof rule for impartial termination. The fairness condition Φ is the above formula. The correctness condition Φ is **false**, which is true only for finite computations.

The Büchi automaton $A_{\Phi,\Psi} = (S, S^0, \alpha, F)$, where $S = \{0, \ldots, n\}$, $S^0 = \{0\}$, $F = \{n\}$ and $(k, i, l) \in \alpha$ if:

- $0 \leq k < n$, $l = k + 1$, and $i \in$ scheduled$_l$, or
- $k = n$ and $l = 0$.

By Theorem 4.4, $P = (W, I, R)$ is impartially terminate iff there exists an ordinal κ and a rank predicate $\rho : 2^{W \times n+1 \times \kappa}$ such that the following holds:

- for all $u \in I$ we have that $\rho(u, 0, \kappa)$ holds,
- for all $u, v \in W$ and $p < n$, if $\rho(u, p, \mu)$ holds, $(u, v) \in R$, and scheduled$_i$ holds at u, then $\rho(v, p + 1, \nu)$ holds for some $\nu \leq \mu$,
- for all $u, v \in W$, if $\rho(u, n, \mu)$ holds and $(u, v) \in R$, then $\rho(v, 0, \nu)$ holds for some $\nu < \mu$.

The reader should compare this rule with Method M for impartial termination in [19]. □

Example 4.6 (*precedence properties*). Precedence properties specify the desired order of events along a computation [23]. A precedence property is expressed by the temporal formula $G(\varphi \rightarrow \varphi_1 \mathcal{U}(\varphi_2 \mathcal{U} \cdots \varphi_n) \cdots))$, where $\mathcal{U}$ is the '*unless*' connective[7]. The property holds for a computation w if for all $i \geq 0$, if $w^i \models \varphi$ there exists a sequence $i = i_1 \leq i_2 \leq \cdots \leq i_n \leq \omega$ such that $w^k \models \varphi_{i_j}$ for $i_j \leq k < i_{j+1}$, $1 \leq j \leq n - 1$, and $w^{i_n} \models \varphi_n$. (We take here the convention that $w^\omega \models \varphi_n$ holds vacuously.)

We now show how to derive a proof rule for precedence properties. We are not concerned here with fairness so the fairness condition Ψ is **true**. Let correctness condition Φ be the above precedence formula. The Büchi automaton $A_{\Phi,\Psi} = (S, S^0, \alpha, F)$, where $S = \{0, \ldots, n, n+1\}$, $S^0 = \{0\}$, $F = \{n+1\}$ and $(k, i, l) \in \alpha$ if:

- $k = l = 0$,
- $k = 0$, $l = \min\{j \mid 1 \leq j < n \text{ and } i \in \varphi_j\}$ is well-defined, $i \in \varphi$ and $i \notin \varphi_n$,
- $k = 0$, $l = n$, $i \in \varphi$ and $i \in \varphi_n$, or
- $k = 0$, $l = n + 1$, $i \in \varphi$ and $i \notin \varphi_j$ for $1 \leq j \leq n$.
- $1 \leq k < n$, $l = \min\{j \mid k \leq j < n \text{ and } i \in \varphi_j\}$ is well-defined and $i \notin \varphi_n$,
- $1 \leq k < n$, $l = n$ and $i \in \varphi_n$,
- $1 \leq k < n$, $l = n + 1$ and $i \notin \varphi_j$ for $k < j \leq n$, or
- $k = l = n + 1$.

[7] $w \models \varphi \mathcal{U} \psi$ if either for all $j \geq 0$ we have $w^j \models \varphi$ or for some $j \geq 0$ we have $w^j \models \psi$ and $w^i \models \varphi$ for $0 \leq i < j$.

Using Theorem 4.4 we can now obtain a proof rule for precedence properties. Details are left to the reader. We note that the obtained rule is considerably more complicated than the rule in [23] that uses the high-level notion of '*leads to*'.

5. Concluding remarks

We have presented an automata-theoretic framework to the verification of concurrent and nondeterministic programs. The basic idea is that to verify that a program P is correct one writes a program A that receives the computation of P as input and diverges only on incorrect computation of P. Now P is correct if and only if the program P_A, which is obtaining by combining P and A, terminates. This unifies previous works on verification of fair termination and verification of temporal properties.

We do not claim that our approach makes verification easy. After all, termination itself is a Π_1^1-complete problem. Rather, the point is that our approach enables one to deal with very complicated correctness conditions by reducing the problem to the most basic one: proving termination.

Acknowledgements

I am grateful to David Harel, Gordon Plotkin and Amir Pnueli for fruitful discussions. In particular, Gordon Plotkin observed that every Σ_1^1 language is definable by a recursive automaton. I would also like to thank David Harel, Joe Halpern and the anonymous referee for helpful comments on a previous draft of this paper.

Note added in proof

A result similar to Theorem 2.1 appeared in: Ph. Darondeau, Separating and testing, Proc. 3rd Symp. on Theoretical Aspects of Computer Science (STACS 86), Lecture Notes in Computer Science 210 (Springer, New York, 1986) 203–212.

References

[1] B. Alpern and F.B. Schneider, Verifying temporal properties without using temporal logic, ACM Trans. Programming Languages 11 (1989) 147–167.

[2] B. Alpern and F.B. Schneider, Proving Boolean combinations of deterministic properties, Proc. 2nd IEEE Symp. on Logic in Computer Science, Ithaca (1987) 131–137.

[3] K.R. Apt, Ten years of Hoare's logic: a survey—Part I, ACM Trans. Programming Languages and Systems 3 (1981) 431–483.
[4] K.R. Apt, Ten years of Hoare's logic: a survey—Part II, Theoret. Comput. Sci. 28 (1984) 83–109.
[5] K.R. Apt, and E.R. Olderog, Proof rules and transformation dealing with fairness, Sci. Comput. Programming 3 (1983) 65–100.
[6] K.R. Apt and G.D. Plotkin, Countable nondeterminism and random assignment, J. Assoc. Comput. Mach. 33 (1986), 724–767.
[7] K.R. Apt, A. Pnueli and J. Stavi, Fair termination revisited—with delay, Theoret. Comput. Sci. 33 (1984) 65–84.
[8] H.J. Boom, A weaker precondition for loops, ACM Trans. Programming Languages and Systems 4 (1982) 668–677.
[9] J.R. Büchi, On a decision method in restricted second-order arithmetics, Proc. Internat. Congress on Logic, Methods and Philosophy of Science 1960 (Stanford Univ. Press, 1962) 1–12.
[10] Y. Choueka, Theories of automata on ω-tapes—a simplified approach, J. Comput. System Sci. 8 (1974) 117–141.
[11] E.M. Clarke, E.A. Emerson and A.P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specification, ACM Trans. Programming Languages and Systems 8 (1986) 244–263.
[12] I. Dayan and D. Harel, Fair termination with cruel schedulers, Fund. Inform. 9 (1986) 1–12.
[13] E. W. Dijkstra, A Discipline of Programming (Prentice-Hall, Englewood Cliffs, NJ, 1976).
[14] N. Francez, Fairness, Texts Monographs Comput. Sci. (Springer, New York, 1986).
[15] O. Grumberg, N. Francez, J.A. Makowsky, and W.P. de Roever, A proof rule for fair termination of guarded command, Inform. and Control 66 (1985) 83–102.
[16] B.T. Hailpern and S.S. Owicki, Modular verification of computer communication protocols, IEEE Trans. Comm. 31 (1983) 56–68.
[17] D. Harel, Effective transformations on infinite trees, with applications to high undecidability, dominoes, and fairness, J. Assoc. Comput. Mach. 33 (1986) 225–248.
[18] D. Harel, A. Pnueli and J. Stavi, Propositional dynamic logic of nonregular programs, J. Comput. System Sci. 26 (1983) 222–243.
[19] D. Lehman, A. Pnueli and J. Stavi, Impartiality, justice, and fairness—the ethic of concurrent termination, Proc. 8th Internat. Colloq. on Automata, Language, and Programming, Lecture Notes in Comput Sci. 115 (Springer, New York, 1981) 264–277.
[20] O. Lichtenstein and A. Pnueli, Checking that finite-state concurrent programs satisfy their linear specification, Proc. 12th ACM Symp. on Principles of Programming Languages (1985) 97–107.
[21] Z. Manna and A. Pnueli, Verification of concurrent programs: the temporal framework, in: The Correctness Problem in Computer Science, R.S. Boyer and J.S. Moore, eds., Internat. Lecture Ser. Comput. Sci. (Academic Press, London, 1981) 215–273.
[22] Z. Manna and A. Pnueli, How to cook a temporal proof system for your pet language, Proc. 10th Symp. on Principles of Programming Languages, Austin, TX (1983) 141–154.
[23] Z. Manna and A. Pnueli, Proving precedence properties: The temporal way, Proc. 10th Internat. Colloq. on Automata Languages and Programming, Lecture Notes in Comput. Sci 154 (Springer, New York, 1983) 491–512.
[24] Z. Manna and A. Pnueli, Adequate proof principles for invariance and liveness of concurrent programs, Sci. Comput. Programming 4 (1984) 257–289.
[25] Z. Manna and A. Pnueli, Specification and verification of concurrent programs by $\forall$-automata, Proc 14th ACM Symp. on Principles of Programming Languages, Munich (1987) 1–12.
[26] M. Nivat, Behaviors of processes and synchronized systems, in: M. Broy and G. Schmidt, eds., Theoretical Foundations of Programming Methodology (Reidel, Dordrecht, 1982) 473–550.
[27] S.S. Owicki and L. Lamport, Proving liveness properties of concurrent programs, ACM Trans. Programming Languages and Systems 4 (1982) 455–495.
[28] D. Park, On the semantics of fair parallelism, Proc. Copenhagen Winter School on Abstract Software Specification (1979), Lecture Notes in Comput. Sci. 86 (Springer, New York, 1979).
[29] D. Park, A predicate transformer for weak fair termination, Proc. 6th IBM Symp. on Math. Foundations of Computer Science, Hakone, Japan (1981).

[30] D. Park, Concurrency and automata on infinite sequences, Proc. 5th GI Conf. on Theoretical Computer Science, Lecture Notes in Comput. Sci. 104 (Springer, New York, 1981) 167–183.
[31] A. Pnueli, The temporal logic of programs, Proc. 18th Symp. on Foundations of Computer Science (1977) 46–57.
[32] A. Pnueli, The temporal semantics of concurrent programs, Theoret. Comput. Sci. 13 (1981) 45–60.
[33] J.P. Queille and J. Sifakis, Fairness and related properties in transition systems—a temporal logic to deal with fairness, Acta Inform. 19 (1983) 195–220.
[34] N. Rescher and A. Urquhart, Temporal Logic (Springer, New York, 1971).
[35] R. Rinat, N. Francez and O. Grumberg, Infinite trees, markings, and well-foundedness, Inform. and Comput. 79 (1988) 131–154.
[36] H. Rogers, Theory of Recursive Functions and Effective Computability (McGraw-Hill, New York, 1967).
[37] A.P. Sistla, Fairness, marked trees, and automata, Unpublished manuscript (1986).
[38] A.P. Sistla, On verifying that a concurrent program satisfies a non-deterministic specification, Technical Report TR 88-378.01.1, GTE Laboratories (1988).
[39] A.P. Sistla and E.M. Clarke, The complexity of propositional linear temporal logics, J. Assoc. Comput. Mach. 32 (1985) 733–749.
[40] A.P. Sistla, M.Y. Vardi and P. Wolper, The complementation problem for Büchi automata with applications to temporal logic, Theoret. Comput. Sci. 49 (1987) 217–237.
[41] R.S. Streett, Propositional dynamic logic of looping and converse, Inform. and Control 54 (1982) 121–141.
[42] M.Y. Vardi, Automatic verification of probabilistic concurrent finite-state programs, Proc. 26th IEEE Symp. on Foundations of Computer Science, Portland (1985) 327–338.
[43] M.Y. Vardi and P. Wolper, An automata-theoretic approach to automatic program verification, Proc. IEEE Symp. on Logic in Computer Science, Cambridge (1986) 332–344.
[44] M.Y. Vardi and P. Wolper, Reasoning about infinite computation paths, IBM Research Report RJ6209, (1988).
[45] K. Wagner and L. Staiger, Recursive ω-languages, Proc. Symp. on Foundation of Computation Theory, Lecture Notes in Comput. Sci. 56 (Springer, New York, 1977) 532–537.
[46] P. Wolper, Temporal logic can be more expressive, Inform. and Control 56 (1983) 72–99.
[47] P. Wolper, Expressing interesting properties of programs in propositional temporal logic, Proc. 13th ACM Symp. on Principles of Programming Languages, St. Petersburg Beach, FL (1986) 184–193.