# From Falsification to Verification

Doron Peled* Amir Pnueli† Lenore Zuck‡

May 31, 2001

**Abstract**

This paper enhances the linear temporal logic model checking process with the ability to automatically generate a deductive proof that the system meets its temporal specification. Thus, we emphasize the point of view that model checking can also be used to *justify* why the system actually works. We show that, by exploiting the information in the graph that is generated during the search for counterexamples, when the search of counterexamples fails, we can generate a fully deductive proof that the system meets its specification.

*Bell Laboratories, 600 Mountain Ave., Murray Hill, NJ 07974, E-mail: doron@research.bell-labs.com
†Dept. of Computer Science, Weizmann Institute of Sciences, E-mail: amir@wisdom.weizmann.ac.il
‡Department of Computer Science, New York University, E-mail: zuck@cs.nyu.edu

# 1 Introduction

*Model checking* [CE81, EC80] is an automatic process for verifying temporal properties of finite state systems. It was first applied to Linear Time Temporal Logic (LTL) properties in [LP85, LPZ85]. Model checking techniques construct a (finite) model that represents the joint computations of the system and the negation of the property to be verified, and apply graph algorithmic techniques to the model to check that no computation of the system satisfies the negation of the property (and violates the property.) In the equivalent (and independently developed) automata-theoretic approach ([VW86, Kur95]) both system and (negation of) property are explicitly represented as automata, and the intersection of the automata is checked for emptiness.

When executions that violate the property exist, at least one is reported and serves as a counterexample for the specification. When the search for counterexamples fails one may conclude that the system satisfies its specification. However, our confidence in such a positive conclusion is tarnished by two possible factors:

- For reasons of complexity and decidability, the system that is checked is often an oversimplification of the actual system, hence, the failure to find counterexamples for the checked system does not necessarily imply that the actual system is fault free. Perhaps we simplified away a source of a problem that will cause the actual system to misbehave.
- There always exists the possibility that the model checker itself contains a bug which causes it to report success while, in fact, even the checked system is still faulty.

Both of these risks may cause us to treat with diffidence a result which purely claims success without providing some supporting evidence, a "witness" or "certificate" that the property does indeed hold over the considered system. This 'proof by lack of counterexample' is the main drawback of the model checking approach; some would even say that model checking is a tool for falsification rather than a tool for verification.

An alternative approach to model checking is *deductive verification* that incrementally constructs proofs until the desired conclusion, a proof the system meets its specification, is obtained. Deductive verification is often manual, and, like all deductive proofs, requires considerable human skill and time. One of its main benefits is that it often explains *why* the system satisfies its specification.

In this paper we enhance the LTL model checking process with the ability to automatically generate a deductive proof that the system meets its LTL specification. Thus, we emphasize the point of view that model checking can also be used to *justify* why the system actually works. We show that, by exploiting the information in the graph that is generated during the search for counterexamples, when the search of counterexamples fails, we can generate a fully deductive proof that the system meets its specification.

Several advantages are gained by a checker that, when the property is invalid, produces a counter example, and when the property is valid automatically produces a proof. For one, the proof can be independently checked (by a theorem prover). If the system checked is a simplification of a more complex system, the proof can help to justify (or refute) why the property holds for the more complex system, and, conceivably, be transformed into a similar proof for that complex system.

Moreover, since the deductive proof is based on the simple axiom system of LTL, it can be checked mechanically via an independent theorem prover. Consequently, the automatic proof can be utilized as a device for debugging the model checker itself.

The automata which represent the system and a tester for the LTL property, are represented in this paper as *Just Discrete Systems* (JDS) [KP00]. The JDS model, which is a variant of Fair Transition System model [MP95], is introduced in Section 2. A JDS is a transition systems that includes a set of *justice requirements*, each being an assertion that should be met by a computation of the system infinitely many times. JDSs correspond to Generalized Büchi automata in the automata theoretic view.

Both the system to be checked and the negation of the property are expressed by JDSs. We construct the synchronous parallel composition of these two JDSs to obtain a new JDS, the computations of which are the system computations that violate the property. If such a computation exists–the new JDS is *feasible*–it provides the desired counterexample. Otherwise, the graph generated by the composition is exploited to provide several alternative deductive proofs for the validity of the property over the system's computations.

The main challenge is how to represent the proof that is implicit in the composition JDS. We would like to represent it in a way that would explain to the user why the property holds for the checked system. We explore two alternatives of representing such a proof. Our first proof system (Section 3) automatically generates a proof based on well-founded ranking.

We then (Section 4) provide an algorithms that generates two temporal logic proof scripts, one that proves that the negation of the property is not satisfiable over the system, and the other proves that the property is valid over the system.

The process of generating the second proof script had been implemented in a new system, ProofProd, currently under construction in NYU. In Section 5 we present a sample output of ProofProd. While the machine generated temporal logic proofs seems to be hard to read, minor heuristics can transform them to proofs that resemble efficient human-generated proofs.

**Related Work:** A preliminary version of this work [PZ01] introduces the concept of generating the proof as a complementary stage of model checking using Generalized Büchi automata and general proof rules. In an independent work, Namjoshi [Nam01] has shown a proof system for the $\mu$-calculus, based on alternating tree automata and parity games. There, LTL is treated a special case CTL* using $\forall$-automata, and fairness is recommended to be incorporated into the property.

# 2 Preliminaries

This section describes *Just Discrete Systems* (JSDs), the computational model for the reactive systems we study here, reviews temporal logic, describes tableaux for JDSs, and shows how to use the tableaux for model checking.

## 2.1 Sequences

Let $V$ be a set of typed variables. A $V$-state is an interpretation of $V$, assigning to each variable $x \in V$ a value in its respective domain. We denote by $s[x]$ the value $s$ assigns to variable $x$. Let $\Sigma_V$ denote the set of all $V$-states. A *sequence over* $V$, or a $V$*-sequence*, is a (possibly infinite)

sequence $\sigma = s_0, s_1, \ldots$ over $\Sigma_V$. We defined the *length of* $\sigma$, $|\sigma|$, to be $\omega$ if $\sigma$ is infinite, and the number of states in $\sigma$ if it is finite.

Given a $V$-sequence $\sigma = s_0, s_1, \ldots$ and some $i$, $0 \leq i < |\sigma|$, we denote by $\sigma^i$ the suffix of $\sigma$ obtained by removing its first $i$ elements. For a state assertion $\varphi$, we say that $i$ *is a* $\varphi$*-position of* $\sigma$ if $s_i$ is a $\varphi$-state (i.e., $s_i \models \varphi$).

## 2.2 Just Discrete Systems

As a computational model for reactive systems, we take the model of *just* (*weakly fair*) *discrete system* (JDS), which is a slight variation on the model of *fair transition system* [MP95].
A JDS $\mathcal{D} : \langle V, \mathcal{O}, \Theta, \rho, \mathcal{J} \rangle$ consists of the following components.

- $V = \{u_1, ..., u_n\}$ : A finite set of typed *system variables*, containing data and control variables. The set of *states* (interpretation) over $V$ is denoted by $\Sigma_V$. When $V$ is clear from the context, we denote $\Sigma_V$ simply by $\Sigma$. Note that $\Sigma$ can be both finite or infinite, depending on the domains of $V$.

- $\mathcal{O} = \{o_1, \ldots, o_n\} \subseteq V$ : A finite set of *observable variables*. These are the variables which the environment can observe.

- $\Theta$ : The *initial condition* – an *assertion* (first-order state formula) characterizing the initial states.

- $\rho$ : A *transition relation* – an assertion $\rho(V, V')$, relating the values $V$ of the variables in state $s \in \Sigma$ to the values $V'$ in a successor state $s' \in \Sigma$. For example, the assertion $x' = x + 1$ corresponds to the assignment $x := x + 1$ and states that the value of variable $x$ in the successor state $s'$ is greater by one than its value in state $s$.

  The state $s'$ is defined to be a $\mathcal{D}$*-successor* of state $s$ if

  $$\langle s, s' \rangle \models \rho(V, V').$$

  That is, $\rho$ evaluates to *true* when we interpret every $x \in V$ as $s[x]$ and every $x'$ as $s'[x]$.

- $\mathcal{J} : \{J_1, \ldots, J_k\}$ : A set of *justice* (*weak fairness*) *requirements*. The justice requirement $J \in \mathcal{J}$ is an assertion, intended to guarantee that every computation contains infinitely many $J$-state (states satisfying $J$).

We require that every state $s \in \Sigma$ has at least one $\mathcal{D}$-successor. This is often ensured by including in $\rho$ the *idling* disjunct $V = V'$ (also called the *stuttering* step). In such cases, every state $s$ is its own $\mathcal{D}$-successor.

### Computations

Let $\mathcal{D}$ be a JDS for which the above components have been identified. We define a *computation* of $\mathcal{D}$ to be an infinite sequence of states

$$\sigma : s_0, s_1, s_2, ...,$$

satisfying the following requirements:

- *Initiality:* $s_0$ is initial, i.e., $s_0 \models \Theta$.
- *Consecution:* For each $j = 0, 1, ...$, the state $s_{j+1}$ is a $\mathcal{D}$-successor of the state $s_j$.
- *Justice:* For each $J \in \mathcal{J}$, $\sigma$ contains infinitely many $J$-positions

We denote by $Comp(\mathcal{D})$ the set of computations of JDS $\mathcal{D}$.

A finite or infinite sequence which satisfies the requirement of consecution is called a *run* of $\mathcal{D}$. A run which satisfies the requirement of initiality is called an *initialized run*.

### Observations and Equivalence of Systems

Let $U \subseteq V$ be a subset of the state variables and $s$ be a $V$-state. We denote by $s\Downarrow_U$ the $U$-state, called the *projection of $s$ on $U$*, which is obtained by restricting the interpretation of variables to the variables in $U$.

For a $V$-state sequence $\sigma$: $s_0, s_1, \ldots$, we denote by $\sigma \Downarrow_U$ the projected $U$-state sequence $\sigma\Downarrow_U : s_0\Downarrow_U, s_1\Downarrow_U, \ldots$. An $\mathcal{O}$-state sequence $\Omega$ is called an *observation* of the JDS $\mathcal{D}$ if $\Omega = \sigma\Downarrow_{\mathcal{O}}$ for some computation $\sigma$ of $\mathcal{D}$. We denote by $Obs(\mathcal{D})$ the set of observations of JDS $\mathcal{D}$.

Systems $\mathcal{D}_1$ and $\mathcal{D}_2$ are *comparable* if they have the same sets of observable variables, i.e., $\mathcal{O}_1 = \mathcal{O}_2$. System $\mathcal{D}_2$, is said to be an *abstraction* of the comparable system $\mathcal{D}_2$, denoted $\mathcal{D}_1 \sqsubseteq \mathcal{D}_2$ if $Obs(\mathcal{D}_1) \subseteq Obs(\mathcal{D}_2)$. The comparable systems $\mathcal{D}_1$ and $\mathcal{D}_2$ are said to be *equivalent*, denoted $\mathcal{D}_1 \sim \mathcal{D}_2$, if their sets of observations are identical. That is, $Obs(\mathcal{D}_1) = Obs(\mathcal{D}_2)$. Note that if $\mathcal{D}_1$ and $\mathcal{D}_2$ are equivalent then each of them is an abstraction of the other.

### Feasibility and Viability of Systems

A JDS $\mathcal{D}$ is said to be *feasible* if $\mathcal{D}$ has at least one computation. $\mathcal{D}$ is defined to be *viable* if any finite initialized run of $\mathcal{D}$ can be extended to a computation.

## 2.3 Operations on JDS's

Systems $\mathcal{D}_1$ and $\mathcal{D}_2$ are *compatible* if $V_1 \cap V_2 = \mathcal{O}_1 \cap \mathcal{O}_2$.

The *synchronous parallel composition* of the compatible systems $\mathcal{D}_1$ and $\mathcal{D}_2$, denoted by $\mathcal{D}_1 \,\|\|\, \mathcal{D}_2$, is given by the JDS $\mathcal{D} = \langle V, \mathcal{O}, \Theta, \rho, \mathcal{J}\rangle$, where

$$\begin{array}{lcl} V & = & V_1 \cup V_2 \\ \mathcal{O} & = & \mathcal{O}_1 \cup \mathcal{O}_2 \\ \Theta & = & \Theta_1 \wedge \Theta_2 \\ \rho & = & \rho_1 \wedge \rho_2 \\ \mathcal{J} & = & \mathcal{J}_1 \cup \mathcal{J}_2 \end{array}$$

**Claim 1** *Let $\mathcal{D} = \mathcal{D}_1 \,\|\|\, \mathcal{D}_2$. Then, a $V$-sequence $\sigma$ is a computation of $\mathcal{D}$ iff $\sigma\Downarrow_{V_1}$ is a computation of $\mathcal{D}_1$ and $\sigma\Downarrow_{V_2}$ is a computation of $\mathcal{D}_2$. Similarly, the $\mathcal{O}$-sequence $\sigma$ is an observation of $\mathcal{D}$ iff $\sigma\Downarrow_{\mathcal{O}_1}$ is an observation of $\mathcal{D}_1$ and $\sigma\Downarrow_{\mathcal{O}_2}$ is an observation of $\mathcal{D}_2$.*

Synchronous parallel composition is mainly used for LTL model checking as will be shown below.

## 2.4 Temporal Logic

Let $\Pi$ be a set of propositions, which can also be viewed as a set of boolean variables. A $\Pi$-state $s$ is an interpretation of these variables, which we can also represent as an element of $2^{\Pi}$, i.e. a subset of $\Pi$, where $p \in s$ iff $s[p]$ is interpreted as 1 (true). We consider here linear time propositional temporal logic formulae over $\Pi$, using the Boolean connectives $\vee$ and $\neg$, and the temporal operators *next-time* $\bigcirc$ and *until* $\mathcal{U}$. Temporal logic formulae are interpreted over infinite sequences of $\Pi$-states in the usual way (see, e.g., [MP91].) Thus, $\sigma \models \varphi$ denotes that the infinite $\Pi$-sequence $\sigma$ *satisfies* the temporal formula $\varphi$. Formula $\varphi$ is said to be *satisfiable* is there exists a ($\Pi$-)sequence $\sigma$ satisfying $\varphi$. Formula $\varphi$ is *valid* if $\sigma \models \varphi$ for every $\Pi$-sequence $\sigma$.

Let $P$ be a JDS whose observables are $\mathcal{O} = \Pi$ and $\varphi$ be a temporal formula over $\Pi$. We say that $\varphi$ is *valid over* $P$ (*$P$-valid*) if every observation of $P$ satisfies $\varphi$.

Other Boolean connectives and Temporal operators ($\Box$, $\Diamond$, $\mathcal{V}$, etc.) can be defined using $\vee$, $\neg$, $\bigcirc$, and $\mathcal{U}$. We assume that all the temporal formulae are given in the *positive normal form*, i.e., with negation applied only to propositions. This can be easily achieved by pushing negation inwards, using the following equivalences:

$$\begin{array}{lcllcl}
\neg\neg\varphi & \equiv & \varphi & \neg \bigcirc \varphi & \equiv & \bigcirc \neg\varphi \\
\neg(\varphi \vee \psi) & \equiv & (\neg\varphi \wedge \neg\psi) & \neg(\varphi \wedge \psi) & \equiv & (\neg\varphi \vee \neg\psi) \\
\neg(\varphi \mathcal{U} \psi) & \equiv & (\neg\varphi)\, \mathcal{V}\, (\neg\psi) & \neg(\varphi \,\mathcal{V}\, \psi) & \equiv & (\neg\varphi) \mathcal{U} (\neg\psi)
\end{array}$$

For an arbitrary formula $\varphi$, we denote by $pos(\varphi)$ the positive form formula which is equivalent to $\varphi$. We introduce the notation $p \Rightarrow q$ as an abbreviation of $\Box(p \rightarrow q)$.

## 2.5 Tableaux and their corresponding JDS's

Let $\varphi$ be a temporal logic formula presented in positive normal form. Let $closure(\varphi)$ be the set of formulae including all the sub-formulae of $\varphi$ and the formulae $\bigcirc \psi$ for every $\mathcal{U}$ and $\mathcal{V}$ sub-formula $\psi$ of $\varphi$. A *$\varphi$-atom* $A$ is a consistent subset of $closure(\varphi)$ that satisfies the following:

1. If $A$ contains a conjunction $p \wedge q$ it must contain both $p$ and $q$.
2. If $A$ contains a disjunction $p \vee q$ it must contain $p$ or $q$.
3. If $A$ contains an until formula $\psi = p \mathcal{U} q$, it must contain either $q$, or $p$ and $\bigcirc \psi$; and
4. If $A$ contains $\psi = p \,\mathcal{V}\, q$, it must contain $q$ and either $p$ or $\bigcirc \psi$.

For an atom $A$, we denote by $\chi(A)$ the conjunction of all formulae contained in $A$. An *atom graph* $G_\varphi = \langle \mathcal{A}, \mathcal{A}_0, E \rangle$ for a formula $\varphi$ consists of:

- A set $\mathcal{A}$ of $\varphi$-atoms.
- A subset $\mathcal{A}_0 \subseteq \mathcal{A}$ of *initial atoms*. It is required that $\varphi \in A$ for every $A \in \mathcal{A}_0$.
- A set of edges $E \subseteq \mathcal{A} \times \mathcal{A}$ connecting atoms within the graph $G$. If $(A, B) \in E$, then it is required that $p \in B$ for every next-formula $\bigcirc p \in A$.

The atom graph $G = \langle \mathcal{A}, \mathcal{A}_0, E \rangle$ is said to be a *tableau* for $\varphi$ if

- $\varphi \rightarrow \bigvee_{A \in \mathcal{A}_0} \chi(A)$.
- $\chi(A) \rightarrow \bigvee_{(A,B) \in E} \bigcirc \chi(B)$ for every $A \in \mathcal{A}$.

We refer to a proposition $p \in \Pi$ or a negation of a proposition as a *literal*. For an atom $A$, we denote by $prop(A)$ the conjunction of all literals contained in $A$.

Let $G = \langle \mathcal{A}, \mathcal{A}_0, E \rangle$ be an atom graph, where $\mathcal{A} = \{A_1, \ldots, A_n\}$. We define $\mathcal{D}_G : \langle V, \mathcal{O}, \Theta, \rho, \mathcal{J} \rangle$, the JDS corresponding to $G$ by:

- $V = \Pi \cup \{\kappa : [1..n]\}$. Thus, we include in $V$ a control variable $\kappa$, ranging over $\{1, \ldots, n\}$.
- $\mathcal{O} = \Pi$.
- $\Theta : \quad \bigvee_{A_i \in \mathcal{A}_0} (\kappa = i \ \wedge \ prop(A_i))$.
- $\rho : \quad \bigvee_{(A_i, A_j) \in E} (\kappa = i \ \wedge \ \kappa' = j \ \wedge \ prop(A_j)')$.
- $\mathcal{J} = \{J_{p\mathcal{U}q} \mid p\mathcal{U}q \in \varphi\}$. Thus, $\mathcal{J}$ contains a justice requirement $J_{p\mathcal{U}q}$ for every *until* formula $p\mathcal{U}q$ contained in $closure(\varphi)$. The justice requirement $J_{p\mathcal{U}q}$ is given by

$$J_{p\mathcal{U}q} : \quad \Big( \bigvee_{q \in A_i} (\kappa = i) \Big) \vee \Big( \bigvee_{p\mathcal{U}q \notin A_i} (\kappa = i) \Big)$$

In the case that $G$ is a tableau for the formula $\varphi$, we say that $\mathcal{D}_G$ is a *temporal tester* for $\varphi$ and denote it by $T_\varphi$.

The following theorem summarizes the properties of a temporal tester:

**Theorem 1** *Let $T_\varphi$ be a temporal tester for the formula $\varphi$ over the propositional variables $\Pi$. Then, a $\Pi$-sequence $\sigma$ is an observation of $T_\varphi$ iff $\sigma \models \varphi$.*

*Furthermore, let $\sigma = s_0, s_1, \ldots$ be a computation of $T_\varphi$, let $i \geq 0$ be position, and $j = s_i[\kappa]$ be the value of $\kappa$ in state $s_i$. Then, $\sigma^i \models \chi(A_j)$.*

## 2.6 Model Checking

Let $P$ be a finite-state JDS whose observables are $\Pi$. The goal of model checking is to establish the $P$-validity of a temporal property $\varphi$. This is accomplished by constructing a JDS whose observations are all the $P$-observations that satisfy $\neg\varphi$. If this JDS has no observations (equivalently, no computations), then $\varphi$ is $P$-valid.

Let $\phi = \neg\varphi$, and let $T_\phi = T_{\neg\varphi}$ be a tester for $\phi$. We define the JDS

$$\mathcal{D}_\phi^P = P \,\|\!|\, T_\phi$$

The following theorem follows immediately from Claim 1 and Theorem 1:

**Theorem 2** *The formula $\varphi$ is $P$-valid iff $\mathcal{D}_\phi^P$ is infeasible.*

Consequently, in order to verify that $\varphi$ is $P$-valid, the process of model checking involves constructing the JDS $\mathcal{D}_\phi^P$ and checking that it is infeasible.

## 3 A Well-Founded Approach to $P$-validity

Let $\varphi$ be a temporal formula, $P$ be a program, and $\phi = \neg\varphi$. We keep $\varphi$, $\phi$, and $P$ fixed for the sequel. We describe how to obtain a deductive proof of the $P$-validity of a property $\varphi$.

The JDS $\mathcal{D}_\phi^P$ can be viewed as a (labelled) directed graph $G_\phi^P = (S, S_0, \tau)$, where $S$ is the set of states of $\mathcal{D}_\phi^P$, $S_0$ is the set of initial states, and $\tau$ is the set of edges connecting states to their immediate successors. We assume that every node in $S$ is reachable from an initial state.

A *well-founded* domain $(\mathcal{W}, \succ)$ consists of a set $\mathcal{W}$ and a total ordering relation $\succ$ over $\mathcal{W}$ such that there are no infinitely decreasing $\prec$ chains $a_0 \succ a_1 \succ \ldots$. A *ranking* function for $\mathcal{D}_\phi^P$ is a function that maps the states of $\mathcal{D}_\phi^P$ into a well-founded domain. Assume that the justice requirements of $\mathcal{D}_\phi^P$ are $\{J_1, \ldots, J_r\}$. Rule WELL, presented in Fig. 1, can be used to prove that the system $\mathcal{D}_\phi^P$ is infeasible.

For a JDS $\mathcal{D}_\phi^P$ with justice requirements $J_1, \ldots, J_r$,

| | | | |
|---|---|---|---|
| assertions | $\varphi_1, \ldots, \varphi_r$ | | |
| a well-founded domain | $(\mathcal{W}, \succ)$, | | |
| and ranking functions | $\delta_1, \ldots, \delta_r : S \to \mathcal{W}$ | | |
| W1. | $\Theta$ | $\rightarrow$ | $\bigvee_{j=1}^r \varphi_j$ |
| W2. | $\rho \wedge \varphi_i$ | $\rightarrow$ | $(\varphi_i' \wedge \delta_i = \delta_i') \vee \bigvee_{j=1}^r (\varphi_j' \wedge \delta_i \succ \delta_j')$ for $i = 1, \ldots, r$ |
| W3. | $\rho \wedge \varphi_i \wedge J_i'$ | $\rightarrow$ | $\bigvee_{j=1}^r (\varphi_j' \wedge \delta_i \succ \delta_j')$ for $i = 1, \ldots, r$ |

$\mathcal{D}_\phi^P$ is infeasible

Figure 1: Rule WELL.

**Theorem 3** WELL *is sound.*

**Proof:** It suffices to show that given $\varphi_1, \ldots, \varphi_r$, $(\mathcal{W}, \succ)$, and $\delta_1, \ldots, \delta_r$ that satisfy the three premises W1-W3, $\mathcal{D}_\phi^P$ is infeasible. Assume to the contrary that $\mathcal{D}_\phi^P$ has a computation $\sigma$: $s_0, s_1, \ldots$. Since $\sigma$ is a computation, $s_0 \models \Theta$. By premise W1, there exists some $i_0 \in \{1, \ldots, r\}$ such that $s_0 \models \varphi_{i_0}$. Denote $d_0 = \delta_{i_0}(s_0) \in \mathcal{W}$.

By premises W2 and W3, there exists an $i_1 \in \{1, \ldots, r\}$ such that $s_1 \models \varphi_{i_1}$ and $d_1 = \delta_{i_1}(s_1) \preceq d_0$, where $d_0 = d_1$ implies $i_1 = i_0$ and $s_{i_1} \not\models J_{i_0}$. Proceeding in this way, we identify an infinite index sequence $i_0, i_1, \ldots$ and an infinite rank sequence $d_0 \preceq d_1 \preceq \ldots$ such that for every $j \geq 0$, $s_j \models \varphi_j$, $d_j = \delta_{i_j}(s_j)$, and $d_j = d_{j+1}$ implies that $i_j = i_{j+1}$ and $s_{j+1} \not\models J_{i_j}$.

Since $\mathcal{W}$ is well-founded, the sequence of ranks cannot decrease infinitely many times. Thus, there exists some *stabilizing position* $c \geq 0$ such that for every $j \geq c$, $d_j = d_c$. It follows that for every $j \geq c$, $i_j = i_c$, and $s_{j+1} \not\models J_{i_c}$. Thus, $\sigma$ violates the justice requirement $J_{i_c}$ and is therefore not a computation of $\mathcal{D}_\phi^P$. ∎

We now describe how to automatically obtain the assertions, well-founded domain, and ranking functions: $\mathcal{W} = \mathbb{N}$ and $\succ$ is the usual $>$ ordering. Let $T_\phi^P$ be the dag obtained from $G_\phi^P$ after its separation to strongly connected components (SCCs). Each $T_\phi^P$-node is either a SCC of $G_\phi^P$, or a single $G_\phi^P$-node that is not part of any SCC. Let $L$ be the set of $T_\phi^P$-nodes that are

*not* in any SCC. For every SCC $\mathcal{C}$, let $\text{UNJUST}_{\mathcal{C}} \subseteq \{1,\ldots,r\}$ be the set of indices of the justice requirements that are not satisfied by any of $\mathcal{C}$'s nodes.

For every $i = 1,\ldots,r$, let $\varphi_i$ be a formula that describes the set of all nodes that belong to $L$ or to an SCC that violates $J_i$:

$$\varphi_i = L \cup \{s : s \in \mathcal{C} \text{ s.t. } i \in \text{UNJUST}_{\mathcal{C}}\}$$

As for the ranking, we let $\delta_1 = \ldots = \delta_r$, and denote it by $\delta$. For all $G^P_\phi$ nodes $s$ that are in $T^P_\phi$'s leaf nodes $\delta(s) = 0$. If $\delta(s)$ is undefined, and $\delta$ is defined for all $T^P_\phi$-successors $s_1,\ldots,s_k$ of $s$, then $\delta(s) = \max_{j=1}^{k} \delta(s_j) + 1$.

**Lemma 1 (Completeness)** *If $\mathcal{D}^P_\phi$ is infeasible, then the procedure above produces assertions, well-founded domain, and ranking that satisfy the premises of* WELL.

**Proof:** Since $\mathcal{D}^P_\phi$ is infeasible, every SCC violates some $J_i$, and therefore any node that is in an SCC satisfies some $\varphi_i$. All nodes that are not in any SCC trivially satisfy all $\varphi_i$s. Hence, every $G^P_\phi$-state satisfies some $\varphi_i$. From the construction it is easy to see that every state leads either to a state with the same ranking (in the same SCC), or to a node with a lower ranking (out of the SCC), thus W2 holds. Assume that $s \models \varphi_i$, $s' \models J_i$, and $(s,s') \in \rho$. If $s \in F$, then obviously $\delta(s) > \delta(s')$. If $s \notin F$, then, since $s \models \varphi_i$, it follows that $s'$ and $s$ are not in the same SCC. Consequently, $\delta(s) > \delta(s')$. ∎

# 4 From Falsification to Verification

This section presents a procedure that exploits the information in $G^P_\phi$ to generate deductive temporal proofs of $\varphi$'s $P$-validity. In particular, we present an algorithm that generates simultaneously two proof scripts, one of $\mathcal{D}^P_\phi$'s infeasibility, and one of $\varphi$'s $P$-validity. Thus, the first proof script establishes the falsification of $\phi$, and the second establishes the verification of $\varphi$.

Let $s \in S$ be a state of $\mathcal{D}^P_\phi$, and assume that the state variables of $P$ are $V^P = \{x_1,\ldots,x_r\}$. Let $\lambda(s) = \bigwedge_{i=1}^{r}(x_i = v_i)$. Note that the only $V^P_\phi$-variable not included in $V^P$ is $\kappa$. We denote by $A_s$ the atom $A_j$ where $j = s[\kappa]$ is the value $s$ assigns to $\kappa$. We define

$$\mathcal{X}(s) = \lambda(s) \ \wedge \ \chi(A_s) \qquad \text{and} \qquad \bar{\chi}(s) = \neg\chi(A_s)$$

The formula $\mathcal{X}(s)$ is the *characteristic formula* of $s$. Its intended meaning is to describe the temporal formulae hold when a computation of $\mathcal{D}^P_\phi$ reaches $s$. The formula $\bar{\chi}(s)$ describes the temporal formulae that hold when an initial run of an infeasible $\mathcal{D}^P_\phi$ reach $s$. Note that $\mathcal{X}(s)$ does not refer to the state variable $\kappa$. However, since $\chi(A_j)$ uniquely identifies $\kappa$ as having the value $j$, this is not necessary.

The properties of the characteristic formulae and of the $\bar{\chi}$s are stated in the following claim, whose proof follows immediately from the definitions and from Theorem 1:

**Lemma 2** *For every computation $\sigma = s_0,\ldots$ of $\mathcal{D}^P_\phi$, and every $i \geq 0$, $\sigma^i \models \mathcal{X}(s_i)$. Similarly, if $\mathcal{D}^P_\phi$ is infeasible, then for every run $\sigma = s_0,\ldots$ of $\mathcal{D}^P_\phi$, and every $i \geq 0$, $\sigma^i \models \bar{\chi}(s_i)$*

An immediate corollary of Lemma 2 is:

**Corollary 1** *$\mathcal{D}_\phi^P$ is infeasible iff $\mathcal{X}(s_0) \to \mathrm{F}$ for every $s_0 \in S_0$. Similarly, $\varphi$ is $P$-valid iff $\lambda(s_0) \Rightarrow \bar{\chi}(s_0)$ for every $s_0 \in S_0$.*

Based on Corollary 1, we describe a procedure that attempts to show that $\mathcal{X}(s) \Rightarrow \mathrm{F}$ (for the first proof script) and that $\lambda(s) \Rightarrow \bar{\chi}(s)$ (for the second proof script) for every state $s \in S$. If $\varphi$ is $P$-valid, our procedure terminates after showing the above for every initial state $s_0 \in S_0$, which, by Corollary 1, achieves our goal. If $\varphi$ is not $P$-valid (and $\mathcal{D}_\phi^P$ is feasible) our procedure terminates while failing to show the above for every $s_0 \in S_0$.

Both proofs are achieved by a chain of temporal formulae that are $P$-valid, and each either follows immediately from the properties of $\mathcal{D}_\phi^P$, or from the previous $P$-validates in the chain. Initially, the proof is empty. It then proceeds according to the procedure in Figure 2.

The procedure in Figure 2 describes how both proof scripts are constructed. The procedure resembles the completeness proof of [LP85, LPZ85], but, while the procedure there is purely semantic (working on the graph), here we give it a syntactic flavor.

(a) For each program justice requirement $J \in \mathcal{J}^P$, add the line

$$\Box \Diamond J$$

to the proof.

(b) For every state $s \in S$, add

$$\mathcal{X}(s) \Rightarrow \bigvee_{s' \in \tau(s)} \bigcirc \mathcal{X}(s')$$

to the proof.

(c) Let all $S$'s nodes be "unmarked". Repeat the following while there are unmarked nodes:

(c.1) If there is an unmarked node all of whose successors are marked, then add the line

| $\mathcal{X}(s) \Rightarrow \mathrm{F}$ | $\lambda(s) \Rightarrow \bar{\chi}(s)$ |
|---|---|
| to the first proof script | to the second proof script |

and mark $s$.

(c.2) If there is an SCC $\mathcal{C}$ of $\mathcal{D}_\phi^P$, all of whose nodes are unmarked, such that all exits from $\mathcal{C}$ lead to marked nodes and for some justice requirement $J \in \mathcal{J}$ no state $\mathcal{C}$ is a $J$-state, add, for every node $s \in \mathcal{C}$, the lines

| $\mathcal{X}(s) \Rightarrow \Box \bigvee_{s' \in \mathcal{C}} \mathcal{X}(s')$<br>$\mathcal{X}(s) \Rightarrow \mathrm{F}$ | $\lambda(s) \Rightarrow \bar{\chi}(s)$ |
|---|---|
| to the first proof script | to the second proof script |

and mark $s$.

(c.3) If no node was marked in either (c.1) or (c.2), report a counter-example and halt.

Figure 2: A Procedure to Generate a Temporal Proof of $P$-validity

**Theorem 4** *If the procedure of Figure 2 terminates with all nodes marked , then for every initial node $s_0$, $\mathcal{X}(s_0) \Rightarrow \mathtt{F}$ and $\lambda(s_0) \Rightarrow \bar{\chi}(s_0)$. Moreover, for every initial node $s_0$, the formulae generated by the first proof script constitute a proof that $\mathcal{X}(s_0) \Rightarrow \mathtt{F}$, and the formulae generated by the second proof script constitute a proof that $\lambda(s_0) \Rightarrow \bar{\chi}(s_0)$.*

**Proof Outline:** Note first that the procedure marks all nodes only if there is no path in $\mathcal{D}_\phi^P$ leading from an initial node to a SCC that satisfies every justice requirement. Thus, the procedure terminates with all nodes marked only when $\mathcal{D}_\phi^P$ is infeasible.

The validity of steps (a) and (b) is immediate. In step (c.1) we are considering (and marking, if possible) a node that does not belong to any SCC from the graph. That is, we are removing state $s$ which, at the time of removal, has all its successors marked. Assume that the successors of $s$ in the original graph $G_\phi^P$ are $s_1, \ldots, s_k$. In step (b) we added to the proof the lines

$$\mathcal{X}(s) \Rightarrow \bigvee_{i=1}^{k} \bigcirc \mathcal{X}(s_i) \tag{1}$$

Since all of the successors are marked, it follows that for each $i = 1, \ldots, k$, the first and second proofs contains the lines $\mathcal{X}(s_i) \Rightarrow \mathtt{F}$ and $\lambda(s_i) \Rightarrow \bar{\chi}(s_i)$ respectively. Combining these lines with formula (1), we conclude $\mathcal{X}(s) \Rightarrow \mathtt{F}$ and $\lambda(s) \Rightarrow \bar{\chi}(s)$.

In step (c.2) we are dealing with an SCC $\mathcal{C}$, all of whose nodes are unmarked, and all of whose exit edges lead to marked nodes. Consider a state $s \in \mathcal{C}$. Viewing again proof line (b) for state $s$, we observe that for every immediate successor $s'$ of $s$ which lies outside of $\mathcal{C}$ the proofs contain lines $\mathcal{X}(s') \Rightarrow \mathtt{F}$ and $\lambda(s') \Rightarrow \bar{\chi}(s')$ respectively. Consequently, we can reduce formula (1) into the formula

$$\mathcal{X}(s) \Rightarrow \bigvee_{(s,s') \in \tau, s' \in \mathcal{C}} \bigcirc \mathcal{X}(s') \tag{2}$$

which can be weakened into

$$\mathcal{X}(s) \Rightarrow \bigvee_{s' \in \mathcal{C}} \mathcal{X}(s') \tag{3}$$

Taking the disjunction of formula (3) over all $s \in \mathcal{C}$, we obtain

$$\Big(\bigvee_{s \in \mathcal{C}} \mathcal{X}(s)\Big) \Rightarrow \bigcirc\Big(\bigvee_{s \in \mathcal{C}} \mathcal{X}(s)\Big) \tag{4}$$

from which, by the axioms of LTL, we can infer

$$\mathcal{X}(s) \Rightarrow \Box\Big(\bigvee_{s \in \mathcal{C}} \mathcal{X}(s)\Big) \tag{5}$$

for every $s \in \mathcal{C}$.

There are two possible reasons why SCC $\mathcal{C}$ was marked. Either it failed to satisfy one of the program justice requirements $J \in \mathcal{J}^P$, or one of the atoms $A_s$ for some $s \in \mathcal{C}$ contained the formula $p\mathcal{U}q$ and no state within $\mathcal{C}$ satisfies $q$.

In the first case, the failure to satisfy $J$ implies that $\mathcal{X}(s) \rightarrow \neg J$ for every $s \in \mathcal{C}$. By formula (5) this implies $\mathcal{X}(s) \Rightarrow \Box \neg J$ which, in view of the line $\Box \Diamond J$ originally placed in the proof leads to $\mathcal{X}(s) \Rightarrow \mathrm{F}$, however, since $\mathcal{X}(s) \leftrightarrow \lambda(s) \wedge \neg \bar{\chi}(s)$, it follows that $\lambda(s) \Rightarrow \bar{\chi}(s)$.

Consider the second case in which all atoms within $\mathcal{C}$ contain the formula $p \mathcal{U} q$ but do not contain the formula $q$. Let $s$ be a node within $\mathcal{C}$. Since $\mathcal{C}$ is strongly connected, $s$ must have an immediate predecessor $\widehat{s} \in \mathcal{C}$. Let $s_1, \ldots, s_k$ be all the nodes which are immediate successors of $\widehat{s}$ such that $q \in A_{s_i}$ for every $i = 1, \ldots, k$. From the construction it can be established that

$$\mathcal{X}(s) \wedge q \Rightarrow \bigvee_{i=1}^{k} \mathcal{X}(s_i) \tag{6}$$

Since no node within $\mathcal{C}$ contains $q$ in its atom, it follows that $s_1, \ldots, s_k$ are outside of $\mathcal{C}$ and, as $\mathcal{C}$ is currently marked, the proofs must contain lines $\mathcal{X}(s_i) \Rightarrow \mathrm{F}$ and $\lambda(s_i) \Rightarrow \bar{\chi}(s_i)$ respectively, for every $i = 1, \ldots, k$. Together with formula (6), this implies $\mathcal{X}(s) \wedge q \Rightarrow \mathrm{F}$. It follows that $\mathcal{X}(s) \Rightarrow (p \mathcal{U} q \wedge \neg q)$ for every $s \in \mathcal{C}$. By formula (5) this implies $\mathcal{X}(s) \Rightarrow \Big(\Box(p \mathcal{U} q) \wedge \Box \neg q\Big)$ which is equivalent to $\mathcal{X}(s) \Rightarrow \mathrm{F}$ and to $\lambda(s) \Rightarrow \bar{\chi}(s)$. ∎

# 5 Experimental Results

We have constructed PROOFPROD, a prototype system that generates temporal proofs as described above. For example, we ran PROOFPROD on the following JDS $P$: $V$ consists of $\mathtt{L} \in \{0, 1, 2\}$ and $p \in \{\mathrm{T}, \mathrm{F}\}$. We use $\mathtt{Lj}$ to denote $\mathtt{L} = j$. $\Theta$ is $\mathtt{L0} \wedge p$; The single justice requirement is $J = \mathtt{L0} \vee \mathtt{L2}$. The transition relation $\rho$ is given by:

$$(\mathtt{L0} \wedge \mathtt{L0}' \wedge p') \vee ((\mathtt{L0} \vee \mathtt{L1}) \wedge \mathtt{L1}' \wedge \neg p') \vee ((\mathtt{L1} \vee \mathtt{L2}) \wedge \mathtt{L2}' \wedge p')$$

The property whose $P$-validity we'd like to establish is $\varphi = \Diamond \Box p$. The atom graph for $\phi = \neg \varphi = \Box \Diamond \neg p$ is described in Fig. 3. There, source-less edges point to initial states, and double circles denote states belonging to the (single) justice set $J_\phi$.

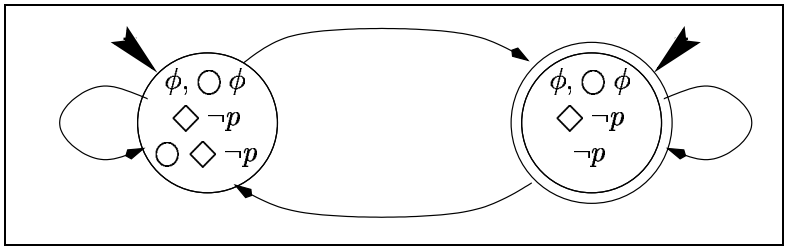


Figure 3: An atom graph of $T_\phi$

Fig. 4 contains the proof that PROOFPROD generated where we used an option that displays the proof in LaTeX.

We are currently working on heuristics to simplify temporal logic formulae that will allow to make the proof more compact. Fairly simple transformations can establish that all of $\Box p$, $\bigcirc \Diamond \Box p$, and $\bigcirc \Box p$ imply $\varphi = \Diamond \Box p$, which helps simplify lines(6)–(9) of the proof in Fig. 4 to

(1) $\Box\Diamond(\mathtt{L0} \vee \mathtt{L2})$
(2) $(\mathtt{L0} \wedge p \wedge \phi \wedge \bigcirc\phi \wedge \Diamond\neg p \wedge \bigcirc\Diamond\neg p) \rightarrow \bigcirc((\mathtt{L0} \wedge p \wedge \phi \wedge \bigcirc\phi \wedge \Diamond\neg p \wedge \bigcirc\Diamond\neg p) \vee$
$(\mathtt{L1} \wedge \neg p \wedge \phi \wedge \bigcirc\phi \wedge \Diamond\neg p \wedge \bigcirc\Diamond\neg p) \vee (\mathtt{L1} \wedge \neg p \wedge \phi \wedge \bigcirc\phi \wedge \Diamond\neg p \wedge \neg p))$
(3) $(\mathtt{L1} \wedge \neg p \wedge \phi \wedge \bigcirc\phi \wedge \Diamond\neg p \wedge \bigcirc\Diamond\neg p) \rightarrow \bigcirc((\mathtt{L1} \wedge \neg p \wedge \phi \wedge \bigcirc\phi \wedge \Diamond\neg p \wedge \bigcirc\Diamond\neg p) \vee$
$(\mathtt{L1} \wedge \neg p \wedge \phi \wedge \bigcirc\phi \wedge \Diamond\neg p \wedge \neg p) \vee (\mathtt{L2} \wedge p \wedge \phi \wedge \bigcirc\phi \wedge \Diamond\neg p \wedge \bigcirc\Diamond\neg p))$
(4) $(\mathtt{L1} \wedge \neg p \wedge \phi \wedge \bigcirc\phi \wedge \Diamond\neg p \wedge \neg p) \rightarrow \bigcirc((\mathtt{L1} \wedge \neg p \wedge \phi \wedge \bigcirc\phi \wedge \Diamond\neg p \wedge \bigcirc\Diamond\neg p) \vee$
$(\mathtt{L1} \wedge \neg p \wedge \phi \wedge \bigcirc\phi \wedge \Diamond\neg p \wedge \neg p) \vee (\mathtt{L2} \wedge p \wedge \phi \wedge \bigcirc\phi \wedge \Diamond\neg p \wedge \bigcirc\Diamond\neg p))$
(5) $(\mathtt{L2} \wedge p \wedge \phi \wedge \bigcirc\phi \wedge \Diamond\neg p \wedge \bigcirc\Diamond\neg p) \rightarrow \bigcirc((\mathtt{L2} \wedge p \wedge \phi \wedge \bigcirc\phi \wedge \Diamond\neg p \wedge \bigcirc\Diamond\neg p))$
(6) $(\mathtt{L2} \wedge p) \rightarrow (\varphi \vee \bigcirc\varphi \vee \Box p \vee \bigcirc\Box p)$
(7) $(\mathtt{L1} \wedge \neg p) \rightarrow (\varphi \vee \bigcirc\varphi \vee \Box p \vee p)$
(8) $(\mathtt{L1} \wedge \neg p) \rightarrow (\varphi \vee \bigcirc\varphi \vee \Box p \vee \bigcirc\Box p)$
(9) $(\mathtt{L0} \wedge p) \rightarrow (\varphi \vee \bigcirc\varphi \vee \Box p \vee \bigcirc\Box p)$

Figure 4: Proof script of $P$-validity of $\varphi$

$(6')$ $(\mathtt{L2} \wedge p) \rightarrow \varphi$ $\quad$ $(7')$ $(\mathtt{L1} \wedge \neg p) \rightarrow (\varphi \vee p)$
$(8')$ $(\mathtt{L1} \wedge \neg p) \rightarrow \varphi$ $\quad$ $(9')$ $(\mathtt{L0} \wedge p) \rightarrow \varphi$

Another set of heuristics is more specific to the type of formulae we obtain in the proof script. Consider the procedure of Figure 2. For every SCC $\mathcal{C}$, let $exit(\mathcal{C}) = \{s' : \text{for some } (s, s') \in \tau, s \in \mathcal{C} \text{ and } s' \notin \mathcal{C}\}$ i.e., $exit(\mathcal{C})$ is the set of all nodes the resider directly outside of $\mathcal{C}$. It is easy to see that, if the procedure terminates when all nodes are marked, then for every SCC C that is marked in step 3.2 because if fails to satisfy a program justice requirement,

$$\lambda(s) \Rightarrow (\bigvee_{t \in \mathcal{C}} \lambda(t) \wedge \bar{\chi}(t)) \mathcal{U} (\bigvee_{t \in exit(\mathcal{C})} \bar{\chi}(t)) \tag{7}$$

Similarly, for every SCC C that is marked in step 3.2 because if fails to satisfy a $p\mathcal{U}q$ justice requirement,

$$\lambda(s) \Rightarrow (\bigvee_{t \in \mathcal{C}} \lambda(t) \wedge \bar{\chi}(t)) \mathcal{W} (\bigvee_{t \in exit(\mathcal{C})} \bar{\chi}(t)) \tag{8}$$

In our example, since the SCC that consists of the single node whose $\lambda$ is $\mathtt{L2} \wedge p$ is removed because if fails to satisfy the justice requirement $\Diamond \neg p$, and its set of exit nodes is empty, we obtain from formula (8) that $\mathtt{L2} \wedge p \Rightarrow (\mathtt{L2} \wedge p \wedge (\varphi \vee p)) \mathcal{W}$ F, which implies (by LTL theorems) and the simplification rules above that $\mathtt{L2} \wedge p \Rightarrow \Box(\mathtt{L2} \wedge p)$. Using a simplification rule "$\Box(p \wedge q) \rightarrow \Box p$", we can obtain $\mathtt{L2} \wedge p \Rightarrow \Box p$ instead of line (6) in the proof.

These, and other heuristics we are working on, help to greatly simplify the proof scripts generated by PROOFPROD, making the proof methodology advocated here both practical and beneficial.

# 6 Conclusion and Future Work

The paper demonstrates how model-checking, that is considered useful only for purposes of falsification, can be used to obtain deductive verification.

As reported in Section 5, we are currently working on heuristics to our system that will generate proofs that are closer to "human" proofs. We are also working on extending our results to apply to systems that employ a wider set of fairness constraints, as well as on obtaining deductive proofs from symbolic model checkers.

**Acknolwedgement** We would like to thank Yi Fang for her technical assistance with PROOF-PROD, and thank Jessie Xu and Yonit Kesten for careful proofreading of the manuscript.

## References

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. IBM Workshop on Logics of Programs*, volume 131 of *Lect. Notes in Comp. Sci.*, pages 52–71. Springer-Verlag, 1981.

[EC80] E.A. Emerson and E.M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proc. 7th Int. Colloq. Aut. Lang. Prog.*, volume 85 of *Lect. Notes in Comp. Sci.*, pages 169–181. Springer-Verlag, 1980.

[KP00] Y. Kesten and A. Pnueli. Verification by finitary abstraction. *Information and Computation, a special issue on Compositionality*, 163:203–243, 2000.

[Kur95] R.P. Kurshan. *Computer Aided Verification of Coordinating Processes.* Princeton University Press, Princeton, New Jersey, 1995.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite-state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symp. Princ. of Prog. Lang.*, pages 97–107, 1985.

[LPZ85] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Proc. Conf. Logics of Programs*, volume 193 of *Lect. Notes in Comp. Sci.*, pages 196–218. Springer-Verlag, 1985.

[MP91] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, New York, 1991.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, New York, 1995.

[Nam01] K.S. Namajoshi. Certifying model checkers. In *Proc.* $13^{rd}$ *Intl. Conference on Computer Aided Verification (CAV'01),* Lect. Notes in Comp. Sci., *Springer-Verlag*, 2001. To appear.

[PZ01] D. Peled and L. Zuck. From model checking to a temporal proof. In *Proc. of the 8th International SPIN Workshop on Model Checking of Software (SPIN'2001)*, volume 2057 of *Lect. Notes in Comp. Sci.*, pages 1–14. Springer-Verlag, 2001.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. First IEEE Symp. Logic in Comp. Sci.*, pages 332–344, 1986.