# Checking That Finite State Concurrent Programs Satisfy Their Linear Specification

Orna Lichtenstein
Tel Aviv University
Ramat Aviv, Tel Aviv, Israel 69978

and

Amir Pnueli
The Weizmann Institute of Science
Rehovot, 76100 Israel

## Abstract

We present an algorithm for checking satisfiability of a linear time temporal logic formula over a finite state concurrent program. The running time of the algorithm is exponential in the size of the formula but linear in the size of the checked program. The algorithm yields also a formal proof in case the formula is valid over the program. The algorithm has four versions that check satisfiability by unrestricted, impartial, just and fair computations of the given program.

## Introduction

Even though there exists a general concensus among a large group of theoreticians and practitioners about the utility and appropriateness of temporal logic as a specification and verification tool for concurrent programs, there is still a major controversy between the advocates of the linear time version and the believers in the branching time version of temporal logic.

Some of the arguments offered by the supporters of the linear time logic ([MP1], [L1], [L2]) are better expressibility, in particular of fairness and liveness properties. Properties that are better expressed by branching time logic such as the ***possibility*** of some computation are claimed by them (us) to be of no interest to the specifiers or verifiers of concurrent systems.

Some of the advantages pointed out by the branching time logic advocates ([CE1], [QS1], [QS2], [EH]) are precisely this ability to express the ***possibility*** of a computation, and representing the branching structure of a computation.

A strong argument in favor of branching time logic has been its better efficiency (lower complexity) for automatic verification. It is not difficult to see that if we restrict our attention to finite state programs, i.e. concurrent programs in which the program variables range over finite domains, then the whole program can be represented as a finite graph consisting of states and transitions connecting the states. Each state contains truth evaluation of a set of atomic properties that can be represented by propositions. Consequently a finite state program can be viewed as a finite model over which ***propositional*** temporal formulas can be evaluated. An important observation is that the majority of communication protocols and examples of distributed and concurrent systems studied in the literature can be represented as finite state programs. The verification of such systems relative to an appropriate propositional temporal logic specification is thus reduced to the problem of ***model-checking***, i.e. checking that a given finite model satisfies a given temporal formula. Apriori this seems an easier problem than the satisfiability problem which is to check whether a given temporal formula is satisfied by any model at all. It is also clear that model-checking as a verification tool subsumes and greatly improves on all the ad-hoc methods using finite state representation for the analysis of protocols. These methods, such as [ZWRCB] usually present separate algorithms for checking different classes of properties such as ***safety, reachability,*** etc. In contrast, an efficient model-checking procedure presents a single algorithm for checking all the properties that can be expressed in temporal logic.

The work in [SC] analyses the complexity of some of the decision problems in both versions of temporal logic. It finds that the satisfiability problem (and hence checking for general validity) is intractable for both versions, being PSPACE-complete for the linear version and EXPTIME-complete for the branching ver-

sion. On the other hand when we consider the more practical model-checking problem we find that for the linear version it is still PSPACE-complete, while for the branching version it is polynomial.

As a result several systems have been constructed to perform model checking of branching-time specifications. These include the systems reported in [CES] and [QS1]. In both cases it was realized that the language is not strong enough to express some of the fairness properties, and attempts to partially repair this were incorporated in [CES] by considering only restricted paths in the model.

In this paper we present a *practical* (so we claim) general algorithm for model-checking of linear time specification, and therefore suggest that efficient model checking procedures is an advantage now shared by both versions, and that the expressibility issue should be the determining factor in preferring one version to the other.

To explain how this claim is still consistent with the lower bound derived in [SC] we should recollect that in model-checking we are given a model $M$ and a formula $\varphi$. Hence the complexity of the decision procedure should refer to $|M|$ and $|\varphi|$, the respective sizes of both objects. Branching time model-checking is indeed polynomial in $|M| \cdot |\varphi|$. The algorithm we present here for linear model-checking runs in worst time $C \cdot |M| \cdot 2^{\alpha|\varphi|}$ for some constants $C$ and $\alpha$. Hence it is still not better than PSPACE-hard in $|\varphi|$ but it is *linear* in $|M|$. Consequently for relatively small $\varphi$'s we obtain a very efficient model-checking procedure. We should remind the readers that the most interesting properties, namely safety, liveness and precedence ([MP1], [MP2]) are expressible by temporal formulas of small size.

Indeed, in [MP2] we presented three separate algorithms for model-checking of properties in the safety, liveness and precedence classes respectively, all of which run in time *linear* in $|M| \cdot |\varphi|$. The present work generalizes these results by presenting a uniform algorithm that reduces to one of these specific algorithms when presented with a formula belonging to the corresponding class.

We believe that the algorithm presented in this paper opens the way to the construction of practical finite state automatic verifiers of linear time temporal logic specifications.

## Programs and Computations

Let $\Pi$ be a set of propositional variables. A concurrent finite state program $P$ over $\Pi$ consists of the following components:

$S$ - A finite set of states $s_1, \ldots, s_n$.

$P_1, \ldots, P_m$ - A finite set of processes. Each process $P_i$: $S \to 2^S$ maps each state $s \in S$ to $P_i(s)$ the set of its *successors* under $P_i$.

$I : S \to 2^{\Pi}$ - An evaluation mapping each state $s$ to the set of propositions $I(s) \subseteq \Pi$ that are true in $s$.

If $P_i(s) \neq \emptyset$ we say that $P_i$ is *enabled* on $s$. A state $s$ is *terminal* if for every $i = 1, \ldots, m$, $P_i(s) = \emptyset$. We augment the set of processes by the *idle* process $P_0$, defined by:

$$P_0(s) = \mathit{if}\ \text{terminal}(s)\ \mathit{then}\{s\}\ \mathit{else}\ \emptyset.$$

This leads to the situation that every state $s$ has some process $P_i$, $i = 0, 1, \ldots, m$ that is enabled on $s$.

A computation of $P$ is an infinite sequence of states

$$\sigma = s_0 \xrightarrow{P_{i_0}} s_1 \xrightarrow{P_{i_1}} s_2 \to \cdots$$

such that for every $j \geq 0$ $s_{j+1} \in P_{i_j}(s_j)$. An $s$-computation of $P$ is a computation $\sigma$ such that $s_0 = s$.

Given a computation $\sigma$ we define for every $k \geq 0$ the $k$-shifted computation:

$$\sigma^{(k)} = s_k \xrightarrow{P_{i_k}} s_{k+1} \xrightarrow{P_{i_{k+1}}} s_{k+2} \to$$

If for some $k = 0, \ldots, m$ there are infinitely many $j$ such that $i_j = k$ then we say that $P_k$ is *activated* infinitely often in $\sigma$. Similarly we say that $P_k$ is *enabled* infinitely often in $\sigma$ if there are infinitely many $j$'s such that $P_k$ is enabled on $s_j$.

Following [LPS] we define:

A computation $\sigma$ is *impartial* if each $P_i$, $i = 1, \ldots, m$ is activated infinitely often in $\sigma$.

A computation $\sigma$ is *just* if for each $i = 1, \ldots, m$ *either* $P_i$ is infinitely often disabled in $\sigma$ *or* $P_i$ is infinitely often activated in $\sigma$.

A computation $\sigma$ is *fair* if for each $i = 1, \ldots, m$ *either* $P_i$ is *continuously* disabled beyond some state $s_j$ in $\sigma$ *or* $P_i$ is infinitely often activated in $\sigma$.

These three definitions represent different versions of fairness that may be required in different frameworks. The algorithm we later present contains a component that varies according to the degree of fairness we would like to impose. Thus by activating the appropriate version we can check satisfiability by an un-

restricted computation or by a computation that is restricted to be impartial, just or fair.

## Specifications and Their Interpretation

We use linear time temporal logic for our specification language ([MP1]). Formulas in the language are constructed over the propositions in $\Pi \cup \{true, false\}$ using the boolean connectives $\neg$ and $\vee$ and the temporal operators $\bigcirc$ (next) and $\mathcal{U}$ (until). Additional boolean connectives (such as $\wedge, \supset, \equiv$) can be defined in the usual way and additional temporal operators can be defined by:

$\Diamond p = true\ \mathcal{U}\ p$ and $\Box p = \neg \Diamond \neg p$.

The truth value of a temporal formula $\varphi$ over a computation $\sigma$ denoted by $\varphi\ |_\sigma$ is inductively defined by:

$$true\ |_\sigma = true, \quad false\ |_\sigma = false$$

For a proposition $Q \in \Pi$, $Q\ |_\sigma = true \Leftrightarrow Q \in I(s_0)$

$\neg\varphi\ |_\sigma = true \Leftrightarrow \varphi\ |_\sigma = false$

$\varphi_1 \vee \varphi_2\ |_\sigma = true \Leftrightarrow \varphi_1\ |_\sigma = true$ or $\varphi_2\ |_\sigma = true$

$\bigcirc\varphi\ |_\sigma = true \Leftrightarrow \varphi\ |_{\sigma(1)} = true$

$\varphi_1 \mathcal{U} \varphi_2\ |_\sigma = true \Leftrightarrow$ For some $k \geq 0$, $\varphi_2\ |_{\sigma(k)} = true$ and for each $j$, $0 \leq j < k$, $\varphi_1\ |_{\sigma(j)} = true$.

If $\varphi\ |_\sigma = true$ we say that $\sigma$ satisfies $\varphi$ and write $\sigma \models \varphi$.

In the following let $\alpha$ denote one of the four classes of unrestricted, impartial, just or fair computations of a program $P$.

If there exists an $\alpha$-computation of $P$ that satisfies $\varphi$ we say that $\varphi$ is $\alpha$-satisfiable in $P$.

If all $\alpha$-computations of $P$ satisfy $\varphi$ we say that $\varphi$ is $\alpha$-valid over $P$.

Obviously $\varphi$ is $\alpha$-valid over $P$ iff $\neg\varphi$ is not $\alpha$-satisfiable in $P$. The algorithm we present below checks whether a formula $\varphi$ is $\alpha$-satisfiable in $P$.

## Closures and Atoms

Let $\varphi$ be a temporal formula. The *closure* of $\varphi$, $CL(\varphi)$ is the smallest set of formulas containing $\varphi$ and satisfying:

$true, false \in CL(\varphi)$

$\neg\psi \in CL(\varphi) \Leftrightarrow \psi \in CL(\varphi)$ (We identify $\neg\neg\psi$ with $\psi$)

$\psi_1 \vee \psi_2 \in CL(\varphi) \Rightarrow \psi_1, \psi_2 \in CL(\varphi)$

$\bigcirc\psi \in CL(\varphi) \Rightarrow \psi \in CL(\varphi)$

$\neg\bigcirc\psi \in CL(\varphi) \Rightarrow \bigcirc\neg\psi \in CL(\varphi)$

$\psi_1\ \mathcal{U}\ \psi_2 \in CL(\varphi) \Rightarrow \psi_1, \psi_2, \bigcirc(\psi_1\ \mathcal{U}\ \psi_2) \in CL(\varphi)$

It can be shown by induction on the structure of $\varphi$ that $|CL(\varphi)| \leq 5|\varphi|$

In checking validity and satisfiability of formulas $\varphi$ over a program $P$, we always assume that the set of propositions $\Pi$ over which $P$ is interpreted by the evaluation $I$ includes all the propositions appearing in $\varphi$.

An *atom* is defined to be a pair $A = (s, F)$ with $s \in S$ a state, and $F \subseteq CL(\varphi) \cup \Pi$ a set of formulas such that:

$true \in F$, $false \notin F$

For each proposition $Q \in \Pi$, $Q \in F \Leftrightarrow Q \in I(s)$.

For every $\psi \in CL(\varphi)$, $\psi \in F \Leftrightarrow \neg\psi \notin F$

For every $\psi_1, \psi_2 \in CL(\varphi)$,
$\psi_1 \vee \psi_2 \in F \Leftrightarrow \psi_1 \in F$ or $\psi_2 \in F$

For every $\neg\bigcirc\psi \in CL(\varphi)$
$\neg\bigcirc\psi \in F \Leftrightarrow \bigcirc\neg\psi \in F$

For every $\psi_1, \psi_2 \in CL(\varphi)$,
$\psi_1\ \mathcal{U}\ \psi_2 \in F \Leftrightarrow \psi_2 \in F$ or $\psi_1, \bigcirc(\psi_1\ \mathcal{U}\ \psi_2) \in F$

For an atom $A$ we denote by $s_A$ and $F_A$ the state and set of formulas comprising $A$.

The set of all atoms is denoted by $At$. Clearly $|At| \leq |S| \cdot 2^{5|\varphi|}$.

We extend the mapping induced by processes to map atoms $P_i : At \to 2^{At}$ by the natural definition:

$$P_i(A) = \{B \in At \mid s_B \in P_i(s_A)\}$$

For an atom $A = (s, F)$ we denote

$$\hat{A} = \bigwedge_{p \in F} p$$

Let $\psi$ be a formula such that $\hat{A} \to \psi$ is an instance of a propositional tautology. Then we write $A \models \psi$.

A formula $\psi$ which is a boolean combination of the propositions in $\Pi$ is called a *state-formula*.

## Fair Paths and Graphs

The procedure for checking satisfiability attempts to construct a structure of atoms that contains an infinite path corresponding to an $\alpha$-computation of $P$ which satisfies $\varphi$.

The constructed structure $\mathcal{A} = (At, R)$ is a graph whose nodes are all the atoms corresponding to $S$ and $\varphi$, and whose edges, given by $R$, are defined by:

$$(A,B)\in R \Leftrightarrow \begin{cases} B\in P_i(A) \text{ for some } i=0,\ldots,m \text{ and} \\ \text{for every formula } \psi,\ \bigcirc\psi\in F_A \Rightarrow \\ \psi\in F_B. \end{cases}$$

We define an $\alpha$-path in $\mathcal{A}$ to be a labeled infinite sequence of atoms,

$$\pi : A_0 \overset{P_{i_0}}{\rightarrow} A_1 \overset{P_{i_1}}{\rightarrow} A_2 \overset{P_{i_2}}{\rightarrow} \cdots \qquad \text{such that:}$$

a) $\pi$ is an infinite path in $\mathcal{A}$ (namely, for every $j\geq 0$, $A_{j+1}\in P_{i_j}(A_j)$ and for every $\bigcirc\psi\in F_{A_j}$ it follows that $\psi\in F_{A_{j+1}}$).

b) The sequence of corresponding states

$$\sigma : s_{A_0} \overset{P_{i_0}}{\rightarrow} s_{A_1} \overset{P_{i_1}}{\rightarrow} s_{A_2} \rightarrow \cdots$$

is an $\alpha$-computation of $P$.

c) For every $j\geq 0$ and for every $\psi_1\, U\, \psi_2\in F_{A_j}$ there exists some $\ell\geq j$ such that $\psi_2\in F_{A_\ell}$.

If $\sigma\models\psi$ we say that the $\alpha$-path $\pi$ ***fulfills*** $\varphi$.

**Proposition 1**

a) An $\alpha$-path $\pi$ fulfills $\varphi$ iff $\varphi\in F_{A_0}$.

b) The formula $\varphi$ is $\alpha$-satisfiable iff there exists an $\alpha$-path $\pi$ fulfilling it.

**Proof:**

a) Let $\pi$ be $A_0 \overset{P_{i_0}}{\rightarrow} A_1 \overset{P_{i_1}}{\rightarrow} A_2 \overset{P_{i_2}}{\rightarrow} \cdots$, and let $\psi\in CL(\varphi)$. The proposition is established by induction on the structure of $\psi$, showing that for every $j\geq 0$ $\psi\in F_{A_j}$ *iff* $\sigma^{(j)}\models\psi$ where $\sigma$ is the computation corresponding to $\pi$.

b) A direct consequence of the definition of an $\alpha$-path is that if there exists an $\alpha$-path fulfilling $\varphi$ then $\varphi$ is $\alpha$-satisfiable by the computation $\sigma$ derived from $\pi$.
For the other direction of part b) assume an $\alpha$-computation $\sigma$ satisfying $\varphi$. It is easy to see that the corresponding $\alpha$-path fulfilling $\psi$ can be defined by:

$$\pi : A_0 \overset{P_{i_0}}{\rightarrow} A_1 \overset{P_{i_1}}{\rightarrow} A_2 \rightarrow \cdots$$

where for every $j\geq 0$, $s_{A_j}=s_j$ and $F_{A_j}=\{\psi\in CL(\varphi)\mid \sigma^{(j)}\models\psi\}\cup I(s_j)$. ∎

From proposition 1 we conclude that if we want to check the $\alpha$-satisfiability of $\varphi$, we have to look for an $\alpha$-path in $\mathcal{A}$, $A_0 \overset{P_{i_0}}{\rightarrow} A_1 \overset{P_{i_1}}{\rightarrow} A_2 \rightarrow \cdots$ such that $\varphi\in F_A$.

Let $\pi$ be an $\alpha$-path in $A$. We denote by $Inf(\pi)$ the subgraph of $A$ consisting of all the atoms that appear infinitely many times in $\pi$. Here and later we specify subgraphs $G$ of $\mathcal{A}$ by defining the set of nodes belonging to $G$ assuming that the edges of $G$ are taken to be all the $\mathcal{A}$-edges that are incident only on $G$-nodes.

It is not difficult to see that for every $\pi$, the subgraph $Inf(\pi)$ is strongly connected.

We give now an independent characterization of such subgraphs dependent on the restriction represented by $\alpha$.

A subgraph $\mathcal{B}\subseteq\mathcal{A}$ is defined to be ***self-fulfilling*** if every atom in $\mathcal{B}$ has at least one outgoing edge and for every atom $A$ in $\mathcal{B}$ and for every $\psi_1\, U\, \psi_2\in F_A$ there exists an atom $B$ in $\mathcal{B}$ such that $\psi_2\in F_B$.

Let $\phi\neq N\subseteq\{1,\ldots,m\}$; A subgraph $\mathcal{B}\subseteq\mathcal{A}$ is called ***impartial with respect*** to $N$ if for every $i\in N$, there exist $A$, $B$ in $\mathcal{B}$ such that $(A,B)\in R$ and $B\in P_i(A)$.

An SCS (Strongly Connected Subgraph) $\mathcal{B}\subseteq\mathcal{A}$ which is self-fulfilling is called an ***unrestricted*** SCS.

An SCS $\mathcal{B}\subseteq\mathcal{A}$ is called an ***impartial***-SCS if it is impartial w.r.t. (with respect to) $\{1,\ldots,m\}$ and is self-fulfilling.

An SCS $\mathcal{B}\subseteq\mathcal{A}$ is called a ***just***-SCS if $\mathcal{B}$ is self-fulfilling and for every $i=1,\ldots,m$ either there exists an atom $A$ in $\mathcal{B}$ such that $P_i(A)=\phi$ or $\mathcal{B}$ is impartial w.r.t. to $\{i\}$.

An SCS $\mathcal{B}\subseteq\mathcal{A}$ is called a ***Fair***-SCS if it is self-fulfilling and for every $i=1,\ldots,m$ either $P_i(A)=\phi$ for every $A$ in $\mathcal{B}$ or $\mathcal{B}$ is impartial w.r.t. to $\{i\}$.

From the above definitions we get:

**Proposition 2**

a) The infinity set of an $\alpha$-path is an $\alpha$-SCS of $\mathcal{A}$.

b) For every $\alpha$-SCS $G\subseteq\mathcal{A}$ there exists an $\alpha$-path $\Pi$ such that $Inf(\Pi)=G$.

**Proof**:

a) Let $G=Inf(\pi)$. As we already observed $G$ is strongly connected. From the definition of the infinity set, it follows that there exists a $k\geq 0$ such that the atoms in $G$ are exactly all the atoms that appear somewhere in

$$\pi^{(k)} : A_k \overset{P_{i_k}}{\rightarrow} A_{k+1} \rightarrow \cdots$$

Let $A \in G$ and $\psi_1 \; U \; \psi_2 \in F_A$. By the comment above $A$ appears in $\pi^{(k)}$, say $A = A_j$ for some $j \geq k$. By condition c of the definition of $\alpha$-paths, there must exist some $\ell \geq j$ such that $\psi_2 \in F_{A_\ell}$. Hence $A_\ell \in G$. It follows that $G$ is self-fulfilling.

This settles immediately claim a) for unrestricted paths $\pi$.

Let $i \in \{1, \ldots, m\}$ be an index such that $\pi$ is impartial with respect to $i$, i.e. there exist infinitely many $j$'s such that $A_j \xrightarrow{P_i} A_{j+1}$ in $\pi$. In particular there exists at least one such $j \geq k$. Therefore, $A_j, A_{j+1} \in G$ and $G$ is impartial with respect to $\{i\}$.

This shows that if $\pi$ is an impartial path, then $G = Inf(\pi)$ is an impartial SCS.

Let $\pi$ be a just path, and $i \in \{1, \ldots, m\}$ some process index.By the definition of just computations, either $\pi$ is impartial with respect to $i$, or there are infinitely many $j$'s such that $P_i(A_j) = \emptyset$. In the first case $G = Inf(\pi)$ is impartial with respect to $\{i\}$. In the second case, by taking $j \geq k$ we are assured of some $A_j \in G$ such that $P_i(A_j) = \emptyset$. Consequently $G$ is a just SCS.

In the case that $\pi$ is a fair path we know that for each $i \in \{1, \ldots m\}$, either $\pi$ is impartial with respect to $i$, or that for some $\ell$, all $j \geq \ell$ satisfy $P_i(A_j) = \emptyset$. In the case that $\pi$ is impartial with respect to $i$, it follows that $G$ is impartial with respect to $\{i\}$. In the other case, we know that each $A \in G$ appears infinitely many times in $\pi$, hence it has some instance $A = A_j$ with $j \geq k$. Consequently for each $A \in G$, $P_i(A) = \emptyset$ as required by the definition of a fair SCS.

b) Let $G$ be an $\alpha$-SCS. Since $G$ is strongly connected it is always possible to construct a finite cyclic path in $G$:

$$\hat{\pi} : A_0 \xrightarrow{P_{i_0}} A_1 \xrightarrow{P_{i_1}} \cdots \rightarrow A_k = A_0$$

such that:

i) Every atom $A \in G$ appears in $\hat{\pi}$.

ii) For every two atoms $A$, $B \in G$ and process $P_j$ such that $B \in P_j(A)$, the transition $A \xrightarrow{P_j} B$ appear somewhere in $\hat{\pi}$.

Take $\pi = \hat{\pi}^\omega$, i.e. the path constructed out of the infinite repetition of $\hat{\pi}$. The path $\pi$ obviously satisfies clause a) of the definition of $\alpha$-paths, and since $G$ is self fulfilling also clause c).

It only remains to check clause b). For the unrestricted case clause b) is trivially true. Let $j$ be an index such that $G$ is impartial with respect to $\{j\}$. By the definition there exist $A$, $B \in G$ such that $B \in P_j(A)$. By the construction of $\pi$ this implies that infinitely many $P_j$-steps are taken in $\pi$, and hence $\pi$ is impartial with respect to $\{j\}$.

It follows that if $G$ is an impartial SCS then $\pi$ is an impartial path.

Assume next that $G$ is a *just* SCS. Thus for each $j \in \{1, \ldots, m\}$, either $G$ is impartial with respect to $\{j\}$ or there exists an $A \in G$ such that $P_j(A) = \emptyset$. Since any such $A$ appears infinitely many times in $\pi$ it follows that $\pi$ is *just* with respect to $j$.

Similarly if $G$ is a fair SCS it follows that for each $j \in \{1, \ldots, m\}$ either $G$ is impartial with respect to $\{j\}$, or *all* atoms $A \in G$ satisfy $P_j(A) = \emptyset$. It follows that $\pi$ is a fair path. ∎

Consequently, instead of searching $\alpha$-paths, which are infinite objects within the structure $A$, it is sufficient to look for $\alpha$-SCC's which are finite subsets of $A$. Thus a consequence of proposition 2 is that a formula $\varphi$ is $\alpha$-satisfiable iff there exists an atom $A$ in $\mathcal{A}$ such that $\varphi \in F_A$ and there exists a path in $\mathcal{A}$ from $A$ to an $\alpha$-SCS.

Unfortunately, there are still too many SCS's within a given graph. Therefore we concentrate on the *maximal* strongly connected components (MSCC's) and examine their $\alpha$-properties. One interesting property is monotonicity: For $\alpha \in$ {unrestricted, impartial, just}, if $G_1$ is an $\alpha$-SCS and $G_1 \subseteq G_2$, where $G_2$ is an SCS, then $G_2$ is also an $\alpha$-SCS.

To see this, let us argue first that the property of being self-fulfilling is monotonic across strongly connected subgraphs. Let $G_1 \subseteq G_2$ and $G_1$ be self-fulfilling. Let $\psi_1 \; U \; \psi_2 \in F_A$ where $A \in G_2$. Since $G_2$ is strongly connected, there exists a finite path $\pi$ leading from $A$ to some $B \in G_1$. By the definition of the $R$ relation connecting atoms in $G$, either for some atom $C \in \pi$, $\psi_2 \in F_C$, or $\psi_1 \; U \; \psi_2 \in B$. In the latter case, since $G_1$ is self-fulfilling, there exists a $D \in G_1$ such that $\psi_2 \in F_D$. In any case we are assured of an atom $E \in G_2$ such that $\psi_2 \in F_E$ ($E$ being $C$ or $D$).

For the other requirements of an $\alpha$-SCS it is clear that they are monotonic.

This, however, is not the case for fair SCS. There we will develop an algorithm for checking whether a maximally strongly connected component *contains* a fair SCS.

Our first algorithm, presented by the following procedure $\alpha$-SAT, checks whether a temporal formula $\varphi$ is $\alpha$-satisfiable over the structure $\mathcal{A}$, where $\alpha \in$ {unrestricted, impartial, just}.

**Procedure** $\alpha$-SAT;
**begin**

Decompose $\mathcal{A}$ into MSCC's;

**repeat**

**if** there is a terminal MSCC (i.e. without edges leading outside the component) which is not an $\alpha$-SCS)

**then** delete it from $\mathcal{A}$

**until** $\mathcal{A} = \emptyset$ **or** every terminal MSCC is an $\alpha$-SCS;

**if** there is an atom $A$ in $\mathcal{A}$ such that $\varphi \in F_A$
**then** report success **else** report failure

**end**

The second algorithm, presented by the following procedure FAIR-SAT, checks whether a temporal formula $\varphi$ is fair-satisfiable over the structure $\mathcal{A}$. Let $\phi \neq N \subseteq \{1, \ldots, m\}$; We say that an SCS $\mathcal{B} \subseteq \mathcal{A}$ is a fair-SCS w.r.t. $N$ if for every $i \in N$ either $P_i(A) = \phi$ for every $A$ in $\mathcal{B}$ or $\mathcal{B}$ is impartial w.r.t. $\{i\}$, and $\mathcal{B}$ is self fulfilling.

The procedure FAIR-SAT uses the recursive boolean function CFAIR $(\mathcal{B}, N)$, where $\mathcal{B} \subseteq \mathcal{A}$ is a SCS and $\phi \neq N \subseteq \{1, \ldots, m\}$, that returns the boolean value *true* if $\mathcal{B}$ contains a Fair-SCS w.r.t. to $N$, and *false* otherwise.

**Procedure** FAIR-SAT;

**begin**

Decompose $\mathcal{A}$ into MSCC's;

**repeat**

**if** there is a terminal MSCC $\mathcal{B}$ such that

CFAIR$(\mathcal{B}, \{1, \ldots, m\}) =$ *false*

**then** delete $\mathcal{B}$ from $\mathcal{A}$

**until** $\mathcal{A} = \emptyset$ **or** every terminal MSCC contains a fair-subgraph;

**If** there is an atom $A$ in $\mathcal{A}$ such that $\varphi \in F_A$

**then** report success **else** report failure

**end**

**function** CFAIR (B,N): **boolean**;
{$\mathcal{B}$ is a strongly connected subgraph, $N \subseteq \{1, \ldots, m\}$ is a set of indices. The procedure checks that $\mathcal{B}$ contains a fair SCS w.r.t. $N$}

**begin**

**if** $\mathcal{B}$ is not self-fulfilling **then return false**;

**if** $\mathcal{B}$ is impartial with respect to $N$

**then return true**;

Let $K = \{j \in N \mid \mathcal{B}$ is not impartial with respect to $\{j\}\}$

Let $\mathcal{B}^K$ be the subgraph obtained from $\mathcal{B}$ by deleting all the atoms $A \in \mathcal{B}$ such that $P_j(A) \neq \emptyset$ for some $j \in K$.

**if** $\mathcal{B}^K = \emptyset$ **then return false**;
Decompose $\mathcal{B}^K$ into maximal strongly connected components $\mathcal{B}_1, \ldots, \mathcal{B}_n$.

**for** $i := 1$ **to** $n$ **do**

**if** CFAIR$(\mathcal{B}_i, N - K)$ **then return true**;

{Otherwise, no component contains a fair SCS w.r.t. $N - K$}

**return false**

**end** {CFAIR}

**Proposition 3**

The temporal formula $\varphi$ is $\alpha$-satisfiable over the structure $\mathcal{A}$ iff the procedure $\alpha$-SAT reports success.

The proof is a direct consequence of the following two claims:

1) $\varphi$ is $\alpha$-satisfiable iff there exists an atom $A$ in $\mathcal{A}$ such that $\varphi \in F_A$ and there exists a path in $\mathcal{A}$ from $A$ to an $\alpha$-SCS.

2) For $\alpha \in$ {unrestricted, impartial, just}: There is an $\alpha$-SCS in $\mathcal{A}$ iff there exists an $\alpha$-MSCC in $A$. This claim is true by the monotonicity of $\alpha$-SCS's. For the fair case: CFAIR$(\mathcal{B}, N)$ is correct, i.e. it reports success iff $\mathcal{B}$ contains a fair-SCS w.r.t. to $N$.

To establish the correctness of the CFAIR procedure, we observe first that it always terminates. This is because the set $K$ is never empty, and hence a procedure called with some $N$ can only issue recursive calls with the corresponding parameter equal to $N - K$, i.e. a set with lower cardinality than $N$.

Next, assume that a call to CFAIR$(\mathcal{B}, N)$ reports success. This can happen only if some recursive invocation of the same procedure, say CFAIR$(\mathcal{B}', N')$ with $\mathcal{B}' \subseteq \mathcal{B}$ and $N' \subseteq N$ reports success because $\mathcal{B}'$ was found to be impartial with respect to $N'$. It is not difficult to see that if $\mathcal{B} \neq \mathcal{B}'$ then for each $j \in N - N'$, $P_j$ is disabled on all the atoms of $\mathcal{B}'$. Thus $\mathcal{B}'$ is fair with respect to $N$. Consequently a report of success implies

the existence of a $B' \subseteq B$ which is a fair SCS with respect to $N$.

Another argument will show that if $B$ does contain an SCS which is fair with respect to $N$, then the CFAIR procedure cannot report failure.

**Proposition 4**

The $\alpha$-validity over $P$ is decidable.

**Proof**: $\varphi$ is $\alpha$-valid over $P$ iff $\neg\varphi$ is not $\alpha$-satisfiable over the structure $\mathcal{A}$ constructed from $P$ and $CL(\varphi)$.

## Complexity of the Checking Algorithm

Let $|\mathcal{A}|$ denote the size of the structure $\mathcal{A}$ which is defined as the number of nodes and edges in $\mathcal{A}$. If we denote by $|P|$ the similar measure for the program $P$, then it is not difficult to bound $|\mathcal{A}|$ by $|\mathcal{A}| \leq |P| \cdot 2^{5|\varphi|}$. We also observe that in the case that each of the processes $P_1, \ldots, P_m$ is deterministic then $|P| \leq (m+1)|S|$ where $|S|$ is the number of states in the program $P$.

As is well known, the decomposition of a graph $G$ into maximal strongly connected components can be accomplished in $O(|G|)$ steps. It is not difficult to see that checking whether a component $B$ is an $\alpha$-SCS for $\alpha =$ ***unrestricted***, ***impartial*** or ***just***, can be done in $O(|B| + m + 5|\varphi|)$. For the fair case, recursion is employed but its depth never exceeds $m$—the number of processes.

Consequently, checking whether a component $B$ contains a ***fair***-SCS can be done in $O(m[|B| + m + 5|\varphi|)]$. We may summarize these arguments to obtain a bound of $O(m \cdot |P| \cdot 2^{5|\varphi|})$ on the time complexity of the algorithm.

## Treating General Fairness

The fairness notions that have been treated so far in this paper are directly related to fairness in scheduling between the different processes that participate in a concurrent system. Fairness, however, may appear in many other contexts providing abstract modelling for diverse phenomena such as: eventually reliable channels, resource allocation, random scheduling, etc. Most of these fairness requirements can be specified by means of a formula of the general form:

$$\Phi : \bigvee_{i \in I} \bigwedge_{j \in J_i} (\Box\Diamond p_j \vee \Diamond\Box q_j)$$

where the index sets $I$ and $J_i$, $i \in I$ are finite and $p_j, q_j$ are state formulas [EL]. To see that all the fairness notions we considered here are special cases of this formula let $e_j$, $j = 1, \ldots, m$ denote the fact that $P_j$ is currently ***enabled***, and let $t_j$ denote the fact that the current step ***taken*** is performed by $P_j$. We actually need an extension of our formalism in order to have these as atomic propositions but the necessary generalization is not difficult.

Then, under these conventions we may express the notions previously considered by:

| | |
|---|---|
| Impartiality: | $\bigwedge_{j=1,\ldots,m}(\Box\Diamond t_j)$ |
| Justice: | $\bigwedge_{j=1,\ldots,m}(\Box\Diamond(t_j \vee \neg e_j))$ |
| Fairness: | $\bigwedge_{j=1,\ldots,m}(\Box\Diamond t_j \vee \Diamond\Box(\neg e_j))$ |

We show now that the algorithm checking fair satisfiability of a formula $\varphi$ can be easily modified to check satisfiability of $\varphi$ under the assumption of generalized fairness as given by the formula $\Phi$.

The procedure FAIR-SAT remains unchanged. For the function CFAIR we observe that the formula $\Phi$ is a disjunction: $\Phi = \bigvee_{i \in I} \Psi_i$ where each $\Psi_i$ has the general form:

$$\Psi = \bigwedge_{j \in J} (\Box\Diamond p_j \vee \Diamond\Box q_j).$$

Thus, in order to check that an SCS $B$ contains an SCS which is fair w.r.t. $\Phi$ it is sufficient to check that it contains an SCS which is fair w.r.t. $\Psi_i$ for some $i \in I$.

Let us consider a formula $\Psi$ of the form above. Let $N \subseteq J$ be a subset of the indices appearing in $\Psi$. An SCS $B$ is defined to be $\Psi$-***impartial*** w.r.t. $N$ if for every $j \in N$, there exists some atom $A \in B$ such that $A \models p_j$.

An SCS $B$ is defined to be $\Psi$-***fair*** if $B$ is self-fulfilling and for each $j \in J$, either ***some*** atom $A \in B$ satisfies $A \models p_j$, or ***all*** atoms $A \in B$ satisfy $A \models q_j$.

It is not difficult to see that if $B$ is a $\Psi$-fair SCS then the path that passes infinitely many times through every atom and every edge of $B$ satisfies $\Psi$. Conversely, a path $\pi$ in $\mathcal{A}$ that satisfies $\Psi$ always defines a $\Psi$-fair SCS by forming $Inf(\pi)$.

The recursive procedure $\Psi$-FAIR$(B, N)$, checks that the component $B$ contains some SCS that is $\Psi$-fair relative to $N \subseteq J$, i.e. is $\Psi^N$-fair, where $\Psi^N$ is obtained from $\Psi$ by replacing $J$ by the smaller set $N$.

**function** $\Psi$-FAIR($\mathcal{B}, N$): **boolean**

{$\mathcal{B}$ a strongly connected subgraph, $N \subseteq J$ is a set of indices. The procedure checks that $\mathcal{B}$ contains a $\psi^N$-fair SCS}

**begin**

**if** $\mathcal{B}$ is not self-fulfilling **then return false**;

**if** $\mathcal{B}$ *is* $\Psi$-impartial w.r.t. $N$ **then return true**;

Let $K = \{j \in N \mid \mathcal{B}$ is not impartial w.r.t. $\{j\}\}$

Let $\mathcal{B}^K$ be the subgraph obtained from $\mathcal{B}$ by deleting all the atoms $A \in \mathcal{B}$ such that $A \models \neg q_j$ for some $j \in K$.

**if** $\mathcal{B}^K = \emptyset$ **then return false**;

Decompose $\mathcal{B}^K$ into maximal strongly connected components $\mathcal{B}_1, \ldots, \mathcal{B}_n$.

**for** $i$ := 1 to $n$ **do**

**if** $\Psi$-FAIR($\mathcal{B}_i, N - K$) **then**

**return true**;

{Otherwise, no $\mathcal{B}_i$ contains a $\Psi^{N-K}$-fair SCS}

**return false**

**end** {$\Psi$-FAIR}

## Completeness of a Deductive Proof System

A general deductive proof system for proving temporal properties of concurrent programs has been presented in [MP3]. In its general form such a system consists of three parts. The *general* part considers only uninterpreted first order temporal formulas. The *domain* part axiomatizes the theory of the domain over which the program may operate. The *program* part axiomatizes the behavior of a specific program. In [MP4] it has been shown that the program part is complete relative to the other two parts. This means that if we may take all the interpreted first order temporal formulas that are true in *all* models as theorems then the full proof system is adequate for proving the validity of such formulas that hold only over models that represent computations of the considered program.

In the case we consider her, that of finite state programs, the general part reduces to its propositional version and no additional domain part is necessary. Since the completeness of the propositional version of the deductive proof system for pure temporal logic (i.e. the general part) has been established (See [SP] for a detailed proof), the relative completeness of [MP4] yields absolute completeness for proving properties of finite state programs.

However, the algorithm presented above provides a constructive direct proof of the completeness of the proof system. This can be summarized by:

**Theorem**

If the algorithm claims that $\varphi$ is unsatisfiable, then $\neg\varphi$ is provable by the proof system described below.

Thus the checking algorithm provides a tool that cuts both ways. For a given $\varphi$ to be checked for validity, it will either produce a model for $\neg\varphi$ or a proof for $\varphi$.

**Proof System – General Part**

The general part consists of the axioms:

A1. $\neg \Diamond p \equiv \Box \neg p$

A2. $\Box(p \to q) \to (\Box p \to \Box q)$

A3. $\Box p \to (p \wedge \bigcirc p \wedge \bigcirc \Box p)$

A4. $\bigcirc \neg p \equiv \neg \bigcirc p$

A5. $\bigcirc(p \to q) \to (\bigcirc p \to \bigcirc q)$

A6. $\Box(p \to \bigcirc p) \to (p \to \Box p)$

A7. $(p \mathcal{U} q) \equiv [q \vee (p \wedge \bigcirc(p \mathcal{U} q))]$

A8. $(p \mathcal{U} q) \to \Diamond q$

It contains the following inference rules:

R1. If $p$ is an instance of a propositional tautology then $\vdash p$.

R2. If $\vdash (p \to q)$ and $\vdash p$, then $\vdash q$.

R3. If $\vdash p$ then $\vdash \Box p$.

**Proof System – Program Part**

The program part is formulated for a specific program $P$ with the usual constituents, $S, P_1, \ldots, P_m$ and $I : S \to 2^{\Pi}$. We assume that $\Pi$ contains a special proposition *at* $s$ for each $s \in S$, such that

$$\begin{array}{ll} & at\ s \in I(s) \ \text{ for every } s \in S \\ \textit{and} & at\ s' \notin I(s) \ \text{ for every } s \neq s', s, s' \in S. \end{array}$$

We define the following formulas:

$$\delta(s) = \bigwedge_{Q \in I(s)} Q \wedge \bigwedge_{Q \in \Pi - I(s)} \neg Q$$

$$e_j = \bigvee_{s \in S, P_j(s) \neq \emptyset} at\ s \ \text{ for } j = 0, 1, \ldots, m.$$

$$t_j = \bigvee_{s' \in P_j(s)} at\ s \wedge \bigcirc(at\ s') \ \text{ for } j = 0, 1, \ldots, m$$

$\delta(s)$ contains all the propositional information that is known about the state $s$. The formula $e_j$ is true if $P_j$ is currently enabled. The formula $t_j$ is true in a sequence if the next step taken is a $P_j$-step.

The program dependent axioms are:

B1. $\bigwedge_{s \neq s'} \neg(at\ s \wedge at\ s')$

B2. $at\ s \rightarrow \delta(s)$

B3. $\bigvee_{j=0}^{m} t_j$

B4. $\bigwedge_{j=1}^{m}(\Box \diamond t_j \vee \diamond \Box(\neg e_j))$

Axiom B1 states that at any instant we may be in at most one state. Axiom B2 states that whenever we are at state $s$ all of $\delta(s)$ holds. Axiom B3 states that some $P_j$ transition ($j = 0, 1, \ldots, m$) is going to follow each instant. Axiom B4 stipulates fairness. Note that B3 implies that at least one $at\ s$ holds at any instant.

### Proof of the Theorem

Because of limitations of space we outline below only the major steps in the proof.

We assume that the algorithm was applied for checking the satisfiability of a formula $\varphi$ and failed. This means that we constructed the structure $\mathcal{A} = (W, R)$ with $W = At$ and decomposed it into maximal strongly connected components $\mathcal{A}_1, \ldots, \mathcal{A}_k$. Then we checked each terminal component for containing a fair SCS and deleted each component for which the test failed. On completion we were either left with an empty graph, or the remaining graph did not contain an atom $A$ such that $\varphi \in F_A$. We wish to show that this implies that $\neg\varphi$ is provable in our system.

The following list of lemmas establish this statement:

L1: For every $\psi \in CL(\varphi)$
$\vdash \psi \rightarrow \bigvee_{A \in W, \psi \in F_A} \hat{A}$

L2: For every $A \in W$
$\vdash \hat{A} \rightarrow \bigcirc(\bigvee_{B \in W,(A,B) \in R} \hat{B})$

The empty disjunction, arising in the case that $A$ has no $R$-successors in $W$, is interpreted as ***false***.

Denote by $R^*$ the reflexive transitive closure of $R$. Then we have:

L3: $\vdash \hat{A} \rightarrow \Box(\bigvee_{B \in W,(A,B) \in R^*} \hat{B})$

In order to follow the deletion process let us denote $W = W_0$ and $R = R_0$. Then at phase $i$ we locate in $W_i$ a terminal component that does not contain a fair SCS and delete it, obtaining $W_{i+1}$ and $R_{i+1}$. Clearly, no more than $k$ phases are required. Hence we will refer to $W_k$ and $R_k$ as the resulting graph after all deletions have been accomplished.

We will prove by induction on $i = 0, 1, \ldots, k$ the following two statements:

L4: For every $A \in W - W_i$, $\vdash \neg \hat{A}$
(i.e. a deleted atom is useless).

L5: For every $A \in W_i$
$\vdash \hat{A} \rightarrow \Box(\bigvee_{B \in W_i,(A,B) \in R_i^*} \hat{B})$

For $i = 0$, L4 is vacuously true, and L5 is indentical to L3. Consider the passage from $i$ to $i+1$. Let $C$ be the deleted component. Then $W_{i+1} = W_i - C$. Since $C$ is a terminal component relative to $W_i$, we have by L5:

L6: For every $A \in C$
$\vdash \hat{A} \rightarrow \Box(\bigvee_{B \in C} \hat{B})$

We proceed to show that for every $A \in C$, $\vdash \neg \hat{A}$. Consider the different reasons for deleting $C$.

a) If $C = \{A\}$ such that $A$ has no $R_i$-successor, then by L6, $\vdash \hat{A} \rightarrow \Box\hat{A}$ which can be shown to contradict B3.

b) If $C$ is not self-fulfilling then for every $A \in C$ there exists a formula $p\ \mathcal{U}\ q \in F_A$ such that for all $B \in C$, $q \notin F_B$. In this case we can prove

$$\vdash \hat{A} \rightarrow [p\ \mathcal{U}\ q \wedge \Box(\neg q)]$$

which by A8 leads to $\models \neg \hat{A}$.

c) If $C$ does not contain a fair SCS we can show that $\Box(\bigvee_{B \in C} \hat{B})$ contradicts B4, leading again to $\vdash \neg \hat{A}$.

Since we assumed that the algorithm failed, we know that every atom $A$ such that $\varphi \in F_A$ does not belong to $W_k$. By L4 we have that for every such atom $\vdash \neg \hat{A}$. Combining this with L1 for $\psi = \varphi$ we obtain $\vdash \neg\varphi$.

## Admitting the Past

Surprisingly enough, the algorithm presented above can be extended with very little effort to include also the past fragment of linear temporal logic. Several recent works (e.g. [KVR], [BK]) have indicated that the past operators improve the specification of safety properties and are very crucial for achieving compositionality in temporal verification.

The past fragment that we introduce here may only refer to a bounded history — starting at the beginning of the computation.

The past operators that we introduce are:

$\bigoplus p$ - There exists an immediately preceding instant and it satisfies $p$.

$p\ S\ q$ - $p$ *since* $q$, $q$ has happened in the past and since then $p$ has been continuously true.

Some derived past operators are

$\bigotimes p =\ \neg \bigoplus \neg p$, **if** there exists a predecessor instant it must satisfy $p$.

$\blacklozenge q =$ **true** $S\ q$, $q$ happened in the past.

$\blacksquare p =\ \neg \blacklozenge \neg p$, $p$ has been continuously true.

The evaluation of temporal formulas $\varphi$ is now given with respect to a computation $\sigma$ as before, and a position $j \geq 0$ within this computation. We denote such an evaluation by $\varphi\ |^j_\sigma$. It is defined for both past and future operators as follows:

$true\ |^j_\sigma = true$ and $false\ |^j_\sigma = false$ for every $\sigma$ and $j \geq 0$.

For a proposition $Q \in \Pi$, $Q\ |^j_\sigma = true \Leftrightarrow Q \in I(s_j)$

$\neg\varphi\ |^j_\sigma = true \quad \Leftrightarrow \varphi\ |^j_\sigma = false$

$\varphi_1 \vee \varphi_2\ |^j_\sigma = true \Leftrightarrow \varphi_1\ |^j_\sigma = true$ or $\varphi_2\ |^j_\sigma = true$

$\bigcirc\varphi\ |^j_\sigma = true \quad \Leftrightarrow \varphi\ |^{j+1}_\sigma = true$

$\varphi\ \mathcal{U}\ \psi\ |^j_\sigma = true \quad \Leftrightarrow$ For some $k \geq j$ $\psi\ |^k_\sigma = true$ and for each $i, j \leq i < k, \varphi\ |^i_\sigma = true$.

$\bigoplus \varphi\ |^j_\sigma = true \quad \Leftrightarrow j > 0$ and $\varphi\ |^{j-1}_\sigma = true$

$\varphi\ S\ \psi\ |^j_\sigma = true \quad \Leftrightarrow$ For some $k \leq j$ $\psi\ |^k_\sigma = true$ and for each $i, k < i \leq j, \varphi\ |^i_\sigma = true$

In defining $CL(\varphi)$, we add the following clauses:

$\bigoplus \psi \in CL(\varphi) \Rightarrow \psi \in CL(\varphi)$

$\psi_1\ S\ \psi_2 \in CL(\varphi) \Rightarrow \psi_1, \psi_2, \bigoplus(\psi_1\ S\ \psi_2) \in CL(\varphi)$.

In defining atoms we add the following clauses to the requirements of $F$:

For every $\psi_1, \psi_2 \in CL(\varphi)$

$\psi_1\ S\ \psi_2 \in F \Leftrightarrow \psi_2 \in F$ or $\psi_1, \bigoplus(\psi_1\ S\ \psi_2) \in F$

In defining the structure $\mathcal{A}$ we redefine the relation $R$ as follows:

$$(A,B) \in R \Leftrightarrow \begin{cases} B \in P_i(A) \text{ for some } i = 0,1,\ldots,m, \\ \text{and} \\ \text{for every } \bigcirc\psi \in CL(\varphi), \\ \quad \bigcirc\psi \in F_A \Leftrightarrow \psi \in F_B, \text{ and} \\ \text{for every } \bigoplus\psi \in CL(\varphi), \\ \quad \psi \in F_A \Leftrightarrow \bigoplus\psi \in F_B \end{cases}$$

An atom $A$ such that $F_A$ does not contain any formula of the form $\bigoplus\psi$ is called *initial*.

To the definition of $\alpha$-paths we add the clause:

d) $A_0$ is initial

It can be shown that clause d) implies the following: For every $j \geq 0$ and $\psi_1\ S\ \psi_2 \in F_{A_j}$ there exists an $\ell \leq j$ such that $\psi_2 \in F_{A_\ell}$.

An atom $B$ in $\mathcal{A}$ is called *accessible* if there exists a finite path leading from some $A \in \mathcal{A}$ to $B$ such that $A$ is initial.

An SCS $G$ of $\mathcal{A}$ is called *accessible* if some atom in $G$ is accessible.

To the definition of a self-fulfilling SCS we add the requirement that it be *accessible*. This requirement should therefore be satisfied by any $\alpha$-SCS, and should be checked wherever self-fulfillment is checked.

The procedures $\alpha$-SAT and FAIR-SAT should be changed so that after all the deletions are complete, the final search is for an *initial* atom $A$ such that $\varphi \in F_A$. The procedures then report success only if such an atom is found.

With these modifications, propositions 1 to 4 remain valid, leading to an algorithm for deciding the $\alpha$-validity of a formula $\varphi$ of full temporal logic, i.e. spanning both past and future.

In a subsequent paper we will present an extended deductive proof system for full temporal logic and prove its completeness by a method that extends the restricted competeness proof presented here.

## References


[BK] H. Barringer and R. Kuiper, A Temporal Logic Specification Method Supporting Hierarchical Development, University of Manchester, (1983).

[BMP] M. Ben-Ari, Z. Manna, A. Pnueli, The Temporal Logic of Branching Time, *Acta Informatica* **20** (1983) pp. 207–226.

[CE] E.M. Clarke, E.A. Emerson, Synthesis of Synchronization Skeletons for Branching Time Temporal Logic, *Proc. of the Workshop on Logic of Programs*, Yorktown Heights, NY, Springer Verlag LNCS Vol. **131** (1982).

[CES] E.M. Clarke, E.A. Emerson and A.P. Sistla, Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications: A Practical Approach, *10th ACM Symposium on Principles of Programming Languages*, Austin, Texas, January 1983.

[EH] E.A. Emerson, J.Y. Halpern, Sometimes and Not Never Revisited: On Branching Time Versus Linear Time, *$10^{th}$ ACM Symposium on Principles of Programming Languages*, Austin, Texas, 1983.

[EL] E.A. Emerson and C.L. Lei, Temporal Model Checking Under Generalized Fairness Constraints, University of Texas, Austin July (1984).

[KVR] R. Koymans, J. Vytopil and W.P. de Roever, Real-time Programming and Asynchronous Message Passing, *$2^{nd}$ ACM Symposium on Principles of Distributed Computing*, Montreal, Canada (1983) 187–197.

[L1] L. Lamport, 'Sometime' is Sometimes 'Not Never" A Tutorial on the Temporal Logic of Programs, *Proc. of the $7^{th}$ Annual ACM Symposium on Principles of Programming Languages*, Jan. 1980.

[L2] L. Lamport, What good is Temporal Logic? Information Processing 1983, *Proc. of the $9^{th}$ IFIP Congress*, R.E.A. Mason Editor, North Holland, pp. 657–668.

[LPS] D. Lehmann, A. Pnueli, J. Stavi, Impartiality, Justice and Fairness: The Ethics of Concurrent Termination, Automata, Languages and Programming, Springer Verlag LNCS **115** (1981) pp. 265–277.

[MP1] Z. Manna, A. Pnueli, Verification of Concurrent Programs: The Temporal Framework, in the Correctness Problem in Computer Science, R.S. Boyer, J.S. Moore Editors, International Lecture Series in Computer Science, Academic Press (1981).

[MP2] Z. Manna, A. Pnueli, Proving Precedence Properties: The Temporal Way, Technical Report CS84-04, The Weizmann Institute February 1984, also a shorter version in Automata, Languages and Programming $10^{th}$ Colloquium Barcelona, July, 1983, Springer Verlag LNCS **154**, pp. 491–512.

[MP3] Z. Manna, A. Pnueli, Verification of Concurrent Programs: A Temporal Proof System, Foundations of Computer Science IV, Distributed Systems: Part 2, Semantics and Logic, J.W. DeBakker, J. Van Leeuwen Editors, Mathematical Centre Tracts **159** Amsterdam 1983, pp. 163–255.

[MP4] Z. Manna, A. Pnueli, How to Cook a Temporal Proof System for Your Pet Language, *Symposium on Principles of Programming Languages*, Austin, Texas (1983).

[QS1] J.P. Quelle, J. Sifakis, Specification and Verification of Concurrent Systems in CESAR, *Proc. of the $5^{th}$ International Symposium on Programming*, 1981.

[QS2] J.P. Quelle, J. Sifakis, Fairness and Related Properties in Transition Systems, IMAG Research Report 292, Grenoble, March 1982.

[SC] A.P. Sistla, E.M. Clarke, The Complexity of Propositional Temporal Logic, *$14^{th}$ ACM Symposium on Theory of Computing*, May 1982, pp. 159–167.

[SP] R. Sherman and A. Pnueli, Semantic Tableau for Temporal Logic, Technical Report, CS81–21, Weizmann Institute of Science, September (1981).

[ZWRCB] P. Zafiropulo, C. West, H. Rudin, D. Cowan, D. Brand, Towards Analyzing and Synthesizing Protocols, *IEEE Transactions on Communications*, Vol. COM-28, No. 4, April 1980, pp. 651–671.