# The Glory of The Past

Orna Lichtenstein
Dept. of Computer Science
Tel Aviv University
Ramat Aviv, Israel

Amir Pnueli
and
Lenore Zuck*
Dept. of Applied Mathematics
The Weizmann Institute of Science
Rehovot, 76100 Israel

## Abstract

An extension of propositional temporal logic that includes operators referring to a bounded past is considered. An exponential time decision procedure and a complete axiomatic system are presented. A suggested normal form leads to a syntactic classification of safety and liveness formulae. The adequacy of temporal logic to modular verification is examined. Finally we present the notion of *$\alpha$-fairness* which is proved to fully capture the behavior of probabilistic finite state programs.

## Introduction

The classical temporal logic systems (see [Pr],[K],[RU], [Bur]) are usually symmetric, including operators for describing both future and past events. The application of temporal logic in computer science for the specification and verification of reactive systems, usually included only the future fragment. Resistance to the inclusion of the past operators was based on the strive for minimality and on the observation implied in [GPSS] that if we restrict the domain to executions of systems that have a definite start point in time, then the past operators do not add any expressive power.

Several recent developments provide convincing indications that enriching temporal logic with the past operators, or an equivalent construct, can prove most advantageous.

A first observation is that many statements that arise naturally in specifications, are easier to express using the past operators. For example, the fact that every $q$ is preceded by a $p$ is easily expressed by the past-augmented formula $\Box(q \rightarrow ⬖p)$. In this formula, $⬖$ is the past version of the future $\Diamond$. See [KVR] for example of such statements.

* The work of this author was supported by the Eshkol Fund.

A more important motivation is the use of past in compositional proof systems. The pioneering work in [OG] indicated that completeness of verification systems for concurrent programs cannot be achieved unless the program is augmented by auxiliary variables that the proof may refer to. In the same work, it is suggested that the most general auxiliary variables are ***history variables*** that record a sequence of selected values that some variables assumed in the past. The idea of maintaining history variables and using them in the proof has been extended to other models of computations, such as messages based systems, where they are usually refered to as ***traces*** ([HO],[MC],[CH],[H],[NGO]). In contrast, advocates of the global approach to temporal verification, such as ([OL],[L],[MP3]) suggested that it is sufficient to allow references to ***locations*** in the program instead of the structurally complex history variables. Indeed, [MP2] shows relative completeness of the location based approach for the first order case, and [LP] shows absolute completeness for the finite state case.

Unfortunately (Fortunately?), when we consider modular verification, it is forbidden when specifying a module to refer to any but its externally observable variables. Mentioning locations within a module is definitely abhorred. The usefulness of locations in program verification can be explained by saying that each location provides a succinct but indirect representation of the history that leads to this location. Hence, the alternative approach which will satisfy the rules required by modularity is to have tools for the efficient ***direct*** characterization of the history.

One such tool is, of course, the ***history variables***. The alternative tool, which we explore in this paper, is the extension of the temporal logic system to include past operators. One of the conclusions offered by the paper is that these tools are indeed equivalent, and preferring one to the other is merely a matter of personal taste. The works in [BK] and [Pn2] illustrate the utility of the past extension of temporal logic for modular verification.

Once we added the past operators into temporal logic we found out that they help to clarify several issues. In particular they contribute to a natural definition of the basic and important notions of safety and liveness properties. They help to clarify the additional specification power that temporal logic has over regular expressions which may be described as adequate only for expressing safety properties. (Of course, the same specification power, i.e., ability to also express liveness properties, is shared by $\omega$-regular expressions, and we must take extended temporal logic [W] in order to make the above statement precise).

Investigation of these issues also lead us to refine the notion of extreme fairness which was first introduced in [Pn3] in order to deal with finite state probabilistic programs. We present a revised version of this fairness and show that it precisely captures all the properties of such programs which hold with probability 1.

We should emphasize again that augmentation by the past operators does not increase the expressive power of temporal logic. On the other hand it provides in many cases a more natural and succinct presentation and verification of specifications. Fortunately, neither does this augmentation increase the complexity of the decision procedures for the logic. The version of the past operators we introduce appeared already in [Pr] and is attributed there to Dana Scott. An identical version appears in [SC] where it is proved that the validity problem for the full logic is $P$-space complete.

## A Temporal Logic that Includes the Past

Let $S$ be a set of states, $\Pi$ be a set of ***propositional variables*** and $I$ an ***evaluation*** $I: S \to 2^{\Pi}$ mapping each state $s \in S$ to the set of propositions $I(s) \subseteq \Pi$ that are true in $s$.

A ***computation*** is a (possibly infinite) sequence of states:

$$\sigma : s_0, s_1, \ldots \qquad s_i \in S$$

Usually a computation is generated by an underlying program, however, in most of our discussions we shall not directly use this fact.

If the computation is finite:

$$\sigma : s_0, s_1, \ldots s_k$$

We define the ***length*** of $\sigma$ to be $|\sigma| = k+1$. Otherwise we write $|\sigma| = \omega$.

We introduce a temporal language over the propositions in $\Pi \cup \{true, false\}$ using the boolean connectives $\neg$ and $\vee$, and the temporal operators $\odot$ (strong ***next***), $\mathcal{U}$ (***until***), $\ominus$ (strong ***previous***) and $\mathcal{S}$ (***since***).

We define a satisfiability relation $\models$ between a temporal formula, a finite or infinite computation $\sigma$, and a position $j$, $0 \leq j < |\sigma|$, within the computation. The satisfiability relation is defined as follows:

$$(\sigma, j) \models true, \qquad (\sigma, j) \not\models false \quad \text{for every } \sigma \text{ and } j < |\sigma|$$

For a proposition $Q \in \Pi$,

$$\begin{array}{lll}
(\sigma, j) \models Q & \textit{iff} & Q \in I(s_j). \\
(\sigma, j) \models \neg\varphi & \textit{iff} & (\sigma, j) \not\models \varphi. \\
(\sigma, j) \models (\varphi_1 \vee \varphi_2) & \textit{iff} & (\sigma, j) \models \varphi_1 \textbf{ or } (\sigma, j) \models \varphi_2. \\
(\sigma, j) \models \odot\varphi & \textit{iff} & j+1 < |\sigma| \textbf{ and } (\sigma, j+1) \models \varphi. \\
(\sigma, j) \models (\varphi \mathcal{U} \psi) & \textit{iff} & \text{For some } k,\ j \leq k < |\sigma|,\ (\sigma, k) \models \psi \\
 & & \text{and for every } i,\ j \leq i < k,\ (\sigma, i) \models \varphi. \\
(\sigma, j) \models \ominus\varphi & \textit{iff} & j > 0 \textbf{ and } (\sigma, j-1) \models \varphi. \\
(\sigma, j) \models (\varphi \mathcal{S} \psi) & \textit{iff} & \text{For some } k,\ 0 \leq k \leq j,\ (\sigma, k) \models \psi \\
 & & \textbf{and } \text{for every } i,\ k < i \leq j,\ (\sigma, i) \models \varphi.
\end{array}$$

Additional boolean connectives (such as $\wedge$, $\rightarrow$, $\equiv$) can be defined in the usual way. Additional temporal operators can be defined by:

$\bigcirc p \equiv \neg\odot\neg p$ (weak ***next***).

$\Diamond p \equiv true\,\mathcal{U} p$ (***eventually p***).

$\Box p \equiv \neg\Diamond\neg p$ (always in the future).

$\circleddash p \equiv \neg\ominus\neg p$ (weak ***previous***).

$\diamondminus p \equiv true\,\mathcal{S} p$ (sometime in the past).

$\boxminus p \equiv \neg\diamondminus\neg p$ (always in the past).

If $(\sigma, j) \models \varphi$, for some $j$, $0 \leq j < |\sigma|$, we say that $\varphi$ is satisfiable in $\sigma$. If there is a computation $\sigma$ such that $\varphi$ is satisfiable in $\sigma$, we say that $\varphi$ is satisfiable.

A formula $\varphi$ such that all computations $\sigma$ and all positions $j$, $0 \leq j < |\sigma|$ satisfy $(\sigma, j) \models \varphi$ is called *valid*.

For example, $\Diamond\ominus$*false* is a valid formula. It states that all computations have a starting point (position 0).

In the following we refer to general formulae in the described language as TL-formulae. Formulae that have no past operators are called TLF-formulae, and those that have no future operators are called TLP-formulae.

## Decidability

As we already mentioned above, the validity problem for the full TL has been solved in [SC]. Sistla and Clarke present in that paper a $P$-space decision procedure for satisfiability and also show that the problem is $P$-space complete. We, however, prefer to outline a different decision procedure that is connected to the completeness proof and has a better time complexity than the naive implementation of the procedure in [SC]. The decision procedure is very similar to the one presented in [LP] for model checking of TL formulae, which is simplified here by removing the constraints imposed by the underlying program.

## A Procedure for Checking Satisfiability

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

$$\begin{array}{lcl}
\textit{true}, \textit{false} \in CL(\varphi) & & \\
\neg\psi \in CL(\varphi) & \Leftrightarrow & \psi \in CL(\varphi) \ \text{(We identify } \neg\neg\psi \text{ with } \psi \text{ and } \neg\textit{true} \text{ with } \textit{false}) \\
\psi_1 \vee \psi_2 \in CL(\varphi) & \Rightarrow & \psi_1, \psi_2 \in CL(\varphi) \\
\odot\psi \in CL(\varphi) & \Rightarrow & \psi \in CL(\varphi) \\
\psi_1 \,\mathcal{U}\, \psi_2 \in CL(\varphi) & \Rightarrow & \psi_1, \psi_2, \odot(\psi_1 \,\mathcal{U}\, \psi_2) \in CL(\varphi) \\
\ominus\psi \in CL(\varphi) & \Rightarrow & \psi \in CL(\varphi) \\
\psi_1 \mathcal{S} \psi_2 \in CL(\varphi) & \Rightarrow & \psi_1, \psi_2, \ominus(\psi_1 \mathcal{S} \psi_2) \in CL(\varphi).
\end{array}$$

It can be shown by induction on the structure of $\varphi$ that $|CL(\varphi)| \leq 4|\varphi|$.

An *atom* is a set of formulae $A \subseteq CL(\varphi)$ such that

$$\begin{array}{lcl}
\textit{true} \in A, & & \\
\text{For every } \psi \in CL(\varphi), \quad \psi \in A & \Leftrightarrow & \neg\psi \notin A \\
\text{For every } \psi_1 \vee \psi_2 \in CL(\varphi), \psi_1 \vee \psi_2 \in A & \Leftrightarrow & \psi_1 \in A \text{ or } \psi_2 \in A \\
\text{For every } \psi_1 \,\mathcal{U}\, \psi_2 \in CL(\varphi), \psi_1 \,\mathcal{U}\, \psi_2 \in A & \Leftrightarrow & \psi_2 \in A \text{ or } \psi_1, \odot(\psi_1 \,\mathcal{U}\, \psi_2) \in A \\
\text{For every } \psi_1 \mathcal{S} \psi_2 \in CL(\varphi), \psi_1 \mathcal{S} \psi_2 \in A & \Leftrightarrow & \psi_2 \in A \text{ or } \psi_1,\ \ominus(\psi_1 \mathcal{S} \psi_2) \in A
\end{array}$$

The set of all atoms is denoted by $At$. Clearly $|At| \leq 2^{4|\varphi|}$.

An atom $A$ that does not contain any formula of the form $\ominus\psi$ is called *initial*. An atom $A$ that does not contain any formula of the form $\odot\psi$ is called *terminal*.

The procedure for checking satisfiability attempts to construct a structure of atoms, interpreted as states, which contains a path satisfying $\varphi$. When interpreting atoms as states we take the natural evaluation $I$ defined by $I(A) = A \cap \Pi$, i.e., the propositions taken to be true in $A$ are all the propositions contained in $A$.

Initially we construct a structure $\mathcal{A} = (At, R)$ which is a graph whose nodes are all the atoms and whose edges are defined by the relation $R$:

$$(A,B) \in R \Leftrightarrow \begin{cases} \text{for every } \odot\psi \in CL(\varphi), \\ \qquad\qquad \odot\psi \in A \Leftrightarrow \ \psi \in B, \text{ and} \\ \text{for every } \ominus\psi \in CL(\varphi), \\ \qquad\qquad \psi \in A \Leftrightarrow \ \ominus\psi \in B \end{cases}$$

We denote by $R^*$ the reflexive transitive closure of $R$.

A path $\pi = A_0, A_1, \ldots$ in $\mathcal{A}$ is said to ***fulfill*** $\varphi$ if:

a) Every $A_i$, $A_{i+1} \in \pi$ are $R$-connected $((A_i, A_{i+1}) \in R)$. This repeats the requirement that $\pi$ be a path in $\mathcal{A}$.

b) $A_0$ is initial.

c) If $\pi$ is finite then its last atom is terminal.

d) For every $j \geq 0$ and every $\psi_1 \mathcal{U} \psi_2 \in A_j$ there exists some $\ell$, $j \leq \ell < |\pi|$ such that $\psi_2 \in A_\ell$.

e) For some $j$, $0 \leq j < |\pi|$, $\varphi \in A_j$.

Proposition 1

The formula $\varphi$ is satisfiable *iff* there exists a path $\pi$ in $\mathcal{A}$ which fulfills $\varphi$.

Proof: Let $\pi$ be a path in $\mathcal{A}$ which fulfills $\varphi$. It is easy to show by induction that for every $A_k \in \pi$ and $\psi \in A_k$, $(\pi, k) \models \psi$. This is true in particular for the $A_j \in \pi$ such that $\varphi \in A_j$ guaranteed by e).

Given any computation $\sigma = s_0, s_1, \ldots$ over an arbitrary set of states, it is easy to associate with it a path $\pi = A_0, A_1, \ldots$ in $\mathcal{A}$ by defining
$A_i = \{\psi \in CL(\varphi) \mid (\sigma, s_i) \models \psi\}$. ∎

Our construction proceeds by successively removing from $\mathcal{A}$ atoms that could never participate in a path fulfilling $\varphi$. For efficiency sake we remove complete strongly connected subgraphs at a time. The removal process can be described by the sequence of successively smaller structures $\mathcal{A}_i = (W_i, R_i)$, $i = 0, 1, \ldots$ defined by taking:

$$\mathcal{A}_0 = \mathcal{A}, \qquad \text{i.e.,} \quad W_0 = At, \quad R_0 = R.$$

the transition from $\mathcal{A}_i$ to $\mathcal{A}_{i+1}$ can be described as follows:

Let $C$ be a maximal strongly connected subgraph (MSCS) of $W_i$. $C$ is defined to be ***initial*** in $W_i$ if it has no incoming $R_i$ edges.

$C$ is defined to be ***terminal*** in $W_i$ if it has no outgoing $R_i$ edges.

$C$ is defined to be ***self-fulfilling*** if for every formula $\psi_1 \mathcal{U} \psi_2 \in A \in C$, there exists an atom $B \in C$ such that $\psi_2 \in B$.

An MSCS $C$ is defined to be ***useless*** in $W_i$ if it falls into one of the following cases:

a) $C$ is initial but contains no initial atom.

b) $C$ is terminal but is not self-fulfilling.

c) $C$ consists of a non-terminal atom that has no $R_i$-successors.

We observe that a fulfilling path in $\mathcal{A}_i$ must start at an initial atom and eventually remain contained in a self-fulfilling MSCS. Hence we obtain the following construction:

Let $C$ be a useless MSCS in $\mathcal{A}_i$. Define $\mathcal{A}_{i+1} = (W_{i+1}, R_{i+1})$ by $W_{i+1} = W_i - C$, $R_{i+1} = R_i \cap (W_{i+1} \times W_{i+1})$, i.e., $R_{i+1}$ is $R_i$ restricted to the atoms remaining in $W_{i+1}$.

<u>Proposition 2</u>

The formula $\varphi$ is satisfiable in $\mathcal{A}_i$ ***iff*** it is satisfiable in $\mathcal{A}_{i+1}$.

This is justified by the previous observation that by removing $C$ we do not damage any fulfilling path in $\mathcal{A}_i$.

When $\mathcal{A}_i$ is empty or contains no useless MSCS's the removal process terminates. Let this terminal structure be $\mathcal{A}_k$.

<u>Proposition 3</u>

The formula $\varphi$ is satisfiable ***iff*** there exists an atom $A \in W_k$ such that $\varphi \in A$.

<u>Proof</u>: If $\varphi$ is satisfiable we had a fulfilling path already in $\mathcal{A}_0$. None of the removals damaged it and consequently it still exists in $\mathcal{A}_k$ including the particular atom $A_j$ which by clause e) of the definition of fulfilling paths contains $\varphi$.

Assume, for the converse direction, that $\varphi \in A \in W_k$. By the absence of useless MSCS's in the $\mathcal{A}_k$ we can construct a path $\pi$ that starts at an initial atom, goes through $A$ and proceeds to a terminal self-fulfilling MSCS $C$. If $C$ consists of a single terminal atom $B$, the path $\pi$ stops there. Otherwise it continues in a periodic loop that traverses each of the atoms of $C$ infinitely many times. It can be shown that in both cases $\pi$ is fulfilling for $\varphi$. ∎

The satisfiability checking procedure outlined above can be executed in time linear in $|At| \cdot |\varphi|$, hence in $O(|\varphi| \cdot 2^{4|\varphi|})$.

## Completeness

We propose the following deductive system for establishing the validity of full TL formulae. Due to the symmetry between the past and future parts, we organized the axioms by having a past and a future clause for each axiom.

**Axioms**

| | Past Clause | Future Clause |
|---|---|---|
| A1. | $\neg \circledequal p \equiv \ominus \neg p$ | $\neg \odot p \equiv \bigcirc \neg p$ |
| A2. | $\circledequal p \rightarrow \ominus p$ | $\odot p \rightarrow \bigcirc p$ |
| A3. | $p \rightarrow \ominus \odot p$ | $p \rightarrow \bigcirc \circledequal p$ |
| A4. | $\ominus(p \rightarrow q) \rightarrow (\ominus p \rightarrow \ominus q)$ | $\bigcirc(p \rightarrow q) \rightarrow (\bigcirc p \rightarrow \bigcirc q)$ |
| A5. | $\neg \diamondminus p \equiv \boxminus \neg p$ | $\neg \Diamond p \equiv \Box \neg p$ |
| A6. | $\boxminus(p \rightarrow q) \rightarrow (\boxminus p \rightarrow \boxminus q)$ | $\Box(p \rightarrow q) \rightarrow (\Box p \rightarrow \Box q)$ |
| A7. | $\boxminus p \rightarrow \ominus p$ | $\Box p \rightarrow \bigcirc p$ |
| A8. | $\boxminus(p \rightarrow \ominus p) \rightarrow (p \rightarrow \boxminus p)$ | $\Box(p \rightarrow \bigcirc p) \rightarrow (p \rightarrow \Box p)$ |
| A9. | $p \mathcal{S} q \equiv q \vee [p \wedge \circledequal(p \mathcal{S} q)]$ | $p \mathcal{U} q \equiv q \vee [p \wedge \odot(p \mathcal{U} q)]$ |
| A10. | $\diamondminus \ominus false$ | $p \mathcal{U} q \rightarrow \Diamond q$ |

**Inference Rules**

R1. For each substitution instance of a propositional tautology $p$, $\vdash p$.
R2. If $\vdash p \rightarrow q$ and $\vdash p$ then $\vdash q$.
R3. If $\vdash p$ then both $\vdash \boxminus p$ and $\vdash \Box p$.

Theorem

The deductive system presented above is sound and complete for propositional TL.

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

Assume that the formula $\varphi$ is valid. Then $\neg\varphi$ is unsatisfiable. Apply the above satisfiability checking procedure to $\neg\varphi$. In the application we construct a sequence of structures $\mathcal{A}_0, \ldots \mathcal{A}_k$ which eventually converges. Since $\neg\varphi$ is unsatisfiable, either $W_k = \emptyset$ or for all atoms $A \in W_k$, $\neg\varphi \notin A$.

For an atom $A$ we denote $\hat{A} = \bigwedge_{\psi \in A} \psi$

We prove the following list of lemmas:

L1:

$$\vdash \bigvee_{B \in W_0} \hat{B}$$

Recall that $W_0 = At$.

L2: For every $B \in W_0$,

$$\vdash \hat{B} \to \left[\circleddash\Big(\bigvee_{(A,B)\in R_0} \hat{A}\Big) \land \bigcirc\Big(\bigvee_{(B,C)\in R_0} \hat{C}\Big)\right]$$

The empty disjunction, arising in the case that $B$ has no $R_0$-predecessors or no $R_0$-successors, is interpreted as *false*.

Denote by $R_0^*$ the reflexive transitive closure of $R_0$. Then from L2 and the axiom A8 we can obtain:

L3:

$$\vdash \hat{B} \to \left[\boxminus\Big(\bigvee_{(A,B)\in R_0^*} \hat{A}\Big) \land \Box\Big(\bigvee_{(B,C)\in R_0^*} \hat{C}\Big)\right]$$

We now wish to generalize L2 and L3 to $R_i$, $i = 1, \dots, k$. We will prove by induction on $i = 0, 1, \dots, k$ the following three statements:

L4: For every atom $B \notin W_i$, $\vdash \neg\hat{B}$
i.e., a removed atom is useless.

L5: For every $B \in W_i$

$$\vdash \hat{B} \to \left[\circleddash\Big(\bigvee_{(A,B)\in R_i} \hat{A}\Big) \land \bigcirc\Big(\bigvee_{(B,C)\in R_i} \hat{C}\Big)\right]$$

L6: For every $B \in W_i$,

$$\vdash \hat{B} \to \left[\boxminus\Big(\bigvee_{(A,B)\in R_i^*} \hat{A}\Big) \land \Box\Big(\bigvee_{(B,C)\in R_i^*} \hat{C}\Big)\right]$$

For $i = 0$, L4 is vacuously true and L5 and L6 follow from L2 and L3 respectively. Consider the passage from $i$ to $i+1$. It is effected by removing a strongly connected component $\mathcal{C}$ from $\mathcal{A}_i$. Consider the three different reasons for $\mathcal{C}$ being declared useless in $\mathcal{A}_i$.

a) $\mathcal{C}$ is initial but contains no initial atom.

Since every atom $A \in \mathcal{C}$ is non initial it contain some formula of the form $\circleddash\psi$. Hence we can prove

L7: $\vdash \hat{A} \to \circleddash \mathit{true}$.

For each $B \in \mathcal{C}$ we may apply the past part of L6 and observe that since $\mathcal{C}$ is initial in $\mathcal{A}_i$ any $A$ such that $(A, B) \in R_i^*$ is necessarily contained in $\mathcal{C}$ and satisfies L7. Hence we obtain:

$$\vdash \hat{B} \to \boxminus\circleddash \mathit{true}$$

Since the right hand side of this implication contradicts $\diamondsuit\ominus false$ of A10 we conclude that for each $B \in \mathcal{C}$

$$\vdash \neg\hat{B}.$$

b) $\mathcal{C}$ is terminal but is not self-fulfilling.

In this case for each $B \in \mathcal{C}$ there exists a formula $\psi_1 \mathcal{U} \psi_2 \in B$ such that no $C \in \mathcal{C}$ contains $\psi_2$. We may then use the future part of L6 to derive

$$\vdash \hat{B} \rightarrow [\psi_1 \mathcal{U} \psi_2 \wedge \Box(\neg\psi_2)]$$

which is equivalent to

$$\vdash \neg\hat{B}.$$

c) $\mathcal{C}$ consists of a non-terminal atom that has no $R_i$-successors.

Let $\mathcal{C} = \{B\}$. Since $B$ is a non-terminal atom it contains some $\odot\psi \in B$. Hence we may derive from the future part of L5:

$$\vdash \hat{B} \rightarrow [\odot\psi \wedge \bigcirc false]$$

which again leads to

$$\vdash \neg\hat{B}.$$

This case splitting established L4 for $W_{i+1}$, and it is easy to utilize this additional information (i.e., that all removed atoms $B$ satisfy $\vdash \neg\hat{B}$) to update L5 and L6 to their $R_{i+1}$ version.

Consider now the final structure $W_k$. Combining L4 for $i = k$ with L1 we obtain:

$$\vdash \bigvee_{B \in W_k} \hat{B}.$$

Since the satisfiability check for $\neg\varphi$ failed we know that for each $B \in W_k$, $\neg\varphi \notin B$, hence $\varphi \in B$ and trivially $\vdash \hat{B} \rightarrow \varphi$. It follows that $\vdash \varphi$. ∎

## Extending TL by Quantification

In [W] Wolper complained that temporal logic is not expressive enough, and suggested an extended language, ETL. He also showed that an equivalent extension can be obtained by allowing quantifiers over propositions.

Since most of the subsequent results hold for both TL and ETL, we consider the same extension and introduce a quantified version of our richer language, where quantification over propositions is allowed. We refer to this language as QTL (QTLF and QTLP denoting the future and past fragments respectively).

The syntax of TL is extended by adding formulae of the form $\exists p\ \varphi$ where $p \in \Pi$. We introduce $\forall p\ \varphi$ as abbreviation for $\neg\exists p(\neg\varphi)$.

Given two evaluations $I,\ I' : S \to 2^{\Pi}$, we say that $I$ and $I'$ differ (at most) in $p$ if for every $s \in S$, $I(s) - \{p\} = I'(s) - \{p\}$. Then the semantics of $\exists p\ \varphi$ is given by:

$$(I, \sigma, j) \models \exists p\ \varphi \Leftrightarrow (I', \sigma, j) \models \varphi \quad \text{for some } I' \text{ such that } I \text{ and } I' \text{ differ in } p.$$

For this definition we had to make the dependence of $\models$ on the evaluation $I$ explicit.

## Regular and Star-Free Expressions over Computations

Let $\Sigma$ denote the set of all propositional formulae over $\Pi$. We define a language of regular expressions over $\Sigma$ using the operators $\neg$ (negation), $+$ (union), ; (concatenation) and $*$ (Kleene closure).

Let $\sigma = \sigma_0, \ldots, \sigma_k$, $\sigma' = \sigma'_0, \ldots, \sigma'_m$. The concatenation of the two sequences is $\sigma;\sigma' = \sigma_0, \ldots, \sigma_k, \sigma'_0, \ldots, \sigma'_m$. In the case that $\sigma$ $(\sigma')$ is empty, we define $\sigma;\sigma'$ to be $\sigma'$ $(\sigma)$.

We define a satisfiability relation between a *finite* (possibly empty) sequence $\sigma$ and a regular expression as follows:

For $a \in \Sigma$, a propositional formula over $\Pi$, $\sigma \models a$ *iff* $\sigma$ is a singleton computation $\sigma = (s)$, and $s \models a$. In evaluating $a$ over $s$, we use $I(s)$ for evaluating the propositions that appear in $a$.

$$\begin{array}{lll}
\sigma \models (\neg\alpha) & \textit{iff} & \sigma \not\models \alpha. \\
\sigma \models (\alpha + \beta) & \textit{iff} & \sigma \models \alpha \text{ or } \sigma \models \beta. \\
\sigma \models (\alpha;\beta) & \textit{iff} & \sigma = \sigma_1;\sigma_2 \text{ with } \sigma_1 \models \alpha \text{ and } \sigma_2 \models \beta. \\
\sigma \models (\alpha^*) & \textit{iff} & \text{either } \sigma \text{ is empty, or } \sigma = \sigma_1;\ldots;\sigma_k \text{ such that } \sigma_i \models \alpha \\
 & & \text{for } i = 1, \ldots, k.
\end{array}$$

Thus every finite computation satisfies $(\mathit{true})^*$.

A regular expression is called ***star-free*** if it does not contain the $*$ operator, except perhaps in the form $\mathit{true}^*$.

## History Variables and Predicates

In the current propositional version it is natural to define a ***history variable*** as the variable $h$, such that in position $j$ of a computation $\sigma$ its value is given by $h\ |^j_\sigma = [s_0, \ldots, s_j]$. That is, $h$ is a ***sequence*** variable that accumulates the ***complete*** history of the computation, up to the current state. The expressive power of a language that uses history variables depends on the type of ***predicates*** we may apply to the history variables. In the current framework, we suggest to use star-free (regular) expressions as the history variables. Consequently, we extend the temporal language by allowing formulae of the type $[\alpha]_H$, where $\alpha$ is a star-free (regular) expression over $\Pi$, as defined above.

The semantics of these formulae is given by:

$$(\sigma, j) \models [\alpha]_H \qquad \textit{iff} \quad (s_0, \ldots, s_j) \models \alpha$$

We denote by $\mathbf{TL_H}$ ($\mathbf{QTL_H}$) the extension of temporal logic by the inclusion of the history predicates.

## Finite State Automata on Finite and Infinite Sequences

There is of course a strong connection between regular expressions and finite state automata. A ***semi-automaton*** is a system consisting of

$Q$ – A finite set of states.
$\Sigma'$ – An alphabet.
$\delta$ – A non deterministic transition function $\delta: Q \times \Sigma' \to 2^Q$.

We can easily extend $\delta$ to map $\delta: Q \times \Sigma'^* \to 2^Q$. A semi-automaton $B = (Q, \Sigma', \delta)$ is defined to be ***counter-free*** if there does not exist a sequence of states $q_1, q_2, \dots q_n \in Q$ for $n > 1$ and a word $w \in \Sigma'^*$, such that $q_{i+1} \in \delta(q_i, w)$ for $i = 1, \dots, n-1$ and $q_1 \in \delta(q_n, w)$.

Semi-automata can be completed to automata by adding an initial state and acceptance criteria. A ***regular*** automaton $A = (Q, \Sigma', \delta, q_0, F)$ is a system consisting of a semi-automaton $(Q, \Sigma', \delta)$ plus:

$q_0 \in Q$ an initial state.
$F \subseteq Q$ an accepting set.

The finite string language accepted by the automaton $A$ is defined by:

$$L(A) = \{w \in \Sigma'^* \mid \delta(s_0, w) \cap F \neq \emptyset\}$$

A regular automaton is defined to be a ***counter free*** automaton if its semi-automaton $(Q, \Sigma', \delta)$ is counter-free.

Finite state automata can also be used to define languages of infinite strings.

An ***$\omega$-regular*** automaton $A = (Q, \Sigma', \delta, q_0, \Omega)$ is a system consisting of a semi-automaton $(Q, \Sigma', \delta)$ plus:

$q_0 \in Q$ an initial state.
$\Omega \subseteq 2^Q$ a family of accepting sets.

Given an infinite word $\sigma = a_0, a_1, \dots \in (\Sigma')^\omega$, a ***run*** of $A$ on $\sigma$, $r_A(\sigma)$ is defined to be an infinite sequence of states $r_A(\sigma) = q_0, q_1, \dots \in Q^\omega$ such that the first state is the initial state $q_0$, and for each $i = 0, 1, \dots$ $q_{i+1} \in \delta(q_i, a_i)$. For a run $r_A(\sigma)$, $\inf(r_A(\sigma)) \subseteq Q$ is the set of states that appear infinitely many times in the sequence $r_A(\sigma)$.

The language of infinite strings that is accepted by $A$ is defined by

$$\begin{aligned} L(A) = \{\sigma \in (\Sigma')^\omega \mid & \text{There exists a run } r_A(\sigma) \\ & \text{such that } \inf r_A(\sigma)) \in \Omega\} \end{aligned}$$

An ***$\omega$-regular*** automaton is defined to be an ***$\omega$-counter-free*** automaton if its semi-automaton $(Q, \Sigma', \delta)$ is counter-free.

## Equivalence of Star-Free Expressions and Temporal Logic

The following two theorems form the main support for most of the results that follow.

Theorem 1

Let $L$ be a language of *finite* computations (i.e., a subset of $S^*$). The following conditions are equivalent:

a. $L$ is definable by a star-free (regular) expression.

b. $L$ is definable by a TLF (QTLF) formula.

c. $L$ is accepted by a counter-free (regular) automaton.

In order to consider the case of infinite computations, we introduce the notion of a ***limit*** of star-free (regular) expressions. For a star-free (regular) expression $\alpha$, we define $lim(\alpha)$ to be the language of all infinite $S$ sequences, such that $\sigma \in lim(\alpha)$ *iff* infinitely many finite prefices of $\sigma$, $\sigma'' < \sigma$, satisfy $\sigma'' \models \alpha$.

Theorem 2

Let $L$ be a language of *infinite* computations (i.e., a subset of $S^\omega$). Then the following conditions are equivalent:

d. $L$ is representable as $\bigcup_{i=1}^{n} lim(\alpha_i) \cap \overline{lim(\beta_i)}$, where $\alpha_i, \beta_i$, $i = 1, \ldots, n$, are star-free (regular) expressions.

The operator of complementation is defined by $\overline{L} = S^\omega - L$.

e. $L$ is definable by a TLF (QTLF) formula.

f. $L$ is accepted by an $\omega$-counter-free ($\omega$-regular) automaton.

The proof of these two theorems is based on many previous results, including [Buc], [MNP], [C], [T] and [GPSS], which, when combined, yield the theorems almost immediately. An independent proof of theorem 2 appears in [Pe]. The regular-QTLF part of theorem 2 has already been observed in [VW].

## Past, Future and History Predicates over Finite Sequences

- For every TLF (QTLF) formula $\varphi$, there exists a TLP (QTLP) formula $\varphi'$, such that for every *finite* sequence of length $j+1$, $(\sigma, 0) \models \varphi$ *iff* $(\sigma, j) \models \varphi'$. That is, $\varphi$ stated from the beginning of $\sigma$ is equivalent to $\varphi'$ stated from the end of $\sigma$.

This translation is based on translating $\varphi$ into a star-free (regular) expression $\alpha$, taking $\alpha^R$ (reverse of $\alpha$) and then retranslating into TL according to Theorem 1 with the substitution of $\ominus$ for $\odot$ and $\mathcal{S}$ for $\mathcal{U}$.

To illustrate this translation, consider $\varphi : p\mathcal{U}q$. Its corresponding star free expression is $\alpha : p^*; q; true^*$. The appearance of $p^*$ does not violate the star-freedom requirement since it is an abbreviation to $\neg(true^*;\neg p;true^*)$. The reversed expression $\alpha^R$ is of course $true^*;q;p^*$. Its obvious TLF translation is $\Diamond(q \wedge \bigcirc \Box p)$. Replacing once more future by past we obtain the TLP formula $\varphi' : \diamondminus(q \wedge \ominus \boxminus p)$. Thus for any finite sequence $\sigma$, $|\sigma| = k+1$,

$$(\sigma, 0) \models p\mathcal{U}q \Leftrightarrow (\sigma, k) \models \diamondminus(q \wedge \ominus\boxminus p)$$

- The converse translation from $\varphi'$ to $\varphi$ also exists.
- This yields immediately the translatability of $\mathrm{TLF_H}$ to TL. Since any star-free (regular) history predicate $[\alpha]_H$ appearing in a $\mathrm{TLF_H}$ formula can be replaced by a TLP (QTLP) formula $\varphi_\alpha$, which expresses the same property of the prefix of the sequence up to this point.

## Past and Future over infinite Sequences

The past operators do not add expressive power to temporal logic. In order to compare formulae involving past and future operators we introduce the notion of ***initial equivalence***. Two formulae $\varphi$ and $\varphi'$ are defined to be initially equivalent if for every infinite computation $\sigma$:

$$(\sigma, 0) \models \varphi \Leftrightarrow (\sigma, 0) \models \varphi'$$

This is the same as requiring that $\ominus false \rightarrow [\varphi \equiv \varphi']$ be valid over all infinite computations. The following lemmas compare the past and future fragments.

- For every TL (QTL) formula $\varphi$ that may contain past operators, there exists a TLF (QTLF) formula $\varphi'$, which is initially equivalent to $\varphi$.

A possible proof to this claim (for the TL case) is to first translate $\varphi$ into the first order theory of linear order, a translation which is readily available from the definition of $\models$. Then we may invoke the translation from this language into TLF as is outlined in [GPSS] . For the QTL case we may use a similar translation into the weak second order theory of the successor (see [Buc], [W] and [T] for details).

We cannot hope to translate TLF formulae completely into TLP formulae, but, we can restrict the number of future operators. This is expressed in the following normal form theorems:

- Every TL (QTL) formula $\varphi$ (over infinite sequences) is initially equivalent to a formula of the form:

$$\bigvee_{i=1}^{n} (\Box\Diamond[\alpha_i]_H \wedge \Diamond\Box[\beta_i]_H)$$

where $\alpha_i, \beta_i$ $(i = 1, \ldots, n)$, are star-free (regular) expressions.

To effect this translation we use first Theorem 2 in order to obtain the normal form:

$$\bigcup_{i=1}^{n} [\lim(\alpha_i) \cap \overline{\lim(\beta_i)}]$$

which is obviously equivalent to the formula above.

- In view of the equivalence between past formulae and star-free (regular) history predicates, every TL (QTL) formula is also initially equivalent to a formula of the form:

$$\bigvee_{i=1}^{n} ( \Box\Diamond(\varphi_i) \wedge \Diamond\,\Box(\psi_i))$$

where $\varphi_i, \psi_i$ $(i = 1, \ldots, n)$, are TLP (QTLP) formulae.

As an example to this translation, let $p$,$q$ be two state formulae which, for simplicity, we assume to be exclusive. The formula $\Box(p \rightarrow p\mathcal{U}q)$ is then equivalent to the normal form formula:

$$\Box((\neg p) \rightarrow (\neg p)\mathcal{S}q) \wedge \Box\Diamond((\neg p)\mathcal{S}q)$$

In this representation we used the *weak since* operator, defined by $p\mathcal{S}q = \boxminus p \vee p\hat{\mathcal{S}}q$. While the normal form did not allow formulae of the form $\Box\varphi$ where $\varphi$ is a past-formula, such a formula is initially equivalent to $\Box\Diamond(\boxminus\varphi)$, which is allowed in the normal form.

## Classification of Safety and Liveness Properties

Observing as before, that for past formulae $\varphi$ and $\psi$, $\Box\varphi$ and $\Diamond\psi$ are initially equivalent to $\Box\Diamond(\boxminus\varphi)$ and $\Box\Diamond(\diamondminus\psi)$ respectively, we may extend the normal form theorem in the following way:

- Every TL (QTL) formula is initially equivalent to a *positive* boolean combination of formulae of the following forms:
  $\Box\varphi$, $\Diamond\psi$, $\Box\Diamond\theta$, $\Diamond\,\Box\pi$ for TLP (QTLP) formulae $\varphi$,$\psi$,$\theta$ and $\pi$.

In view of several recent attempts ([AS],[S]) to give characterization of safety and liveness properties, we suggest the following classification:

A formula represents a safety property *iff* it is initially equivalent to a formula of the form $\Box\varphi$ for some past formula $\varphi$.

A formula is a *simple* liveness property *iff* it is *not* a safety property and it is representable in one of the forms $\Diamond\psi$, $\Box\Diamond\psi$, $\Diamond\Box\psi$ for some past formula $\psi$.

A liveness property is any positive boolean combination of simple liveness properties, which is not a safety property.

For example, the formula $\Box(p \rightarrow p\mathcal{U}q)$ (for exclusive $p$ and $q$) is representable as the conjunction of a safety formula $\Box((\neg p) \rightarrow (\neg p)\mathcal{S}q)$ and a simple liveness property $\Box\Diamond((\neg p)\mathcal{S}q)$. On the other hand, it is also representable as a single simple liveness formula $\Box\Diamond[(\neg p)\mathcal{S}q \wedge \boxminus((\neg p) \rightarrow (\neg p)\mathcal{S}q)]$.

Comparing our characterization of safety and liveness to the ones given in [AP] we observe the following:

1. The notions of safety presented here and the one defined in [AP] coincide.
2. The class of liveness properties according to [AP] is strictly contained in our class of liveness properties.

Thus, for example, we classify the property $p\mathcal{U}q$ as a simple liveness property since it is representable as $\Diamond(q \wedge \ominus\boxminus p)$. According to [AP] $p\mathcal{U}q$ is neither a safety nor a liveness property.

Our classification is mainly motivated by the two proof principles that have been advocated in [MP1] and [MP3] for establishing safety and liveness properties respectively. In the presence of the past operators, these proof principles can be considerably strengthened and their adequacy for establishing safety and liveness properties made more apparent.

Consider an underlying program $P$ which produces the computations we wish to specify. Let $\varphi$ and $\psi$ be two TLP (QTLP) formulae. We say that $P$ ***leads from*** $\varphi$ ***to*** $\psi$ (denoted by $\varphi \hookrightarrow \psi$) if for every $\sigma$, a computation of $P$, and every position $0 \leq j < |\sigma| - 1$:

$$(\sigma, j) \models \varphi \Longrightarrow (\sigma, j+1) \models \psi.$$

Then we have the following two rules:

**(Safety)**
Let $\varphi$ be a past formula (TLP, QTLP)

$$\frac{\varphi \hookrightarrow \varphi}{\varphi \rightarrow \Box\varphi}$$

**(Liveness)**
Let $\varphi$ and $\psi$ be past formulae (TLP, QTLP)

$$\frac{\begin{array}{l}\varphi \hookrightarrow (\varphi \vee \psi)\\ \text{Eventually } \ \varphi \hookrightarrow \psi\end{array}}{\varphi \rightarrow \Diamond\psi}$$

The "Eventually $\varphi \hookrightarrow \psi$" premise usually corresponds to some fairness requirement that guarantees an eventual execution of a step that leads from $\varphi$ to $\psi$. The basic liveness rule is developed into the general liveness rule by the addition of explicit well founded induction.

Note that the safety rule with a TLP formula $\varphi$ now covers all the different cases of ***unless*** formulae that previously required separate rules. Similarly the liveness rule covers the ***until*** case and many others.

## Adequacy of TL for Compositional Verification

In the earlier works on the use of TL for verification, we find a style of proof that can be described as ***global.***

The proof reasons about the complete program and formulates assertions that unrestrictedly refer to all variables and parts of the program. In particular, a heavy use is made of location pointers that represent the control site in each module and process. It has been shown in [LP] that for the finite state case the proof system for TL presented in [MP1] is complete for global verification.

However, a very important recent trend represented in [BKP] and [NGO], insists on ***compositional*** or ***modular*** proof style. In this style when proving a property of $S_1 \| S_2$, we must derive first the formulae $\varphi_1$ and $\varphi_2$ that give a partial specification of $S_1$ and $S_2$ respectively, and then infer $\varphi$ from the conjunction of $\varphi_1$ and $\varphi_2$. The principle of modularity forbids $\varphi_1$ to mention any but the variables which are externally observable. In particular $\varphi_1$ should not mention the location pointers, since they are obviously unobservable outside of $S_1$.

An important question is therefore whether the proof system and the language are equal to the task.

Consider the following CSP-like example:

$$\begin{gathered} S_1 :: *[[\ell_0 : a! \rightarrow \ell_c : \text{ Critical Section}]; [\ell_1 : b! \rightarrow \textbf{skip}\ ]\ ] \\ \| \\ S_2 :: *[[m_0 : a? \rightarrow \textbf{skip}]; [m_1 : b? \rightarrow m_c : \text{ Critical Section}]\ ] \end{gathered}$$

The property that we wish to establish is

$$\varphi : \Box\neg(\textit{at } \ell_c \wedge \textit{at } m_c)$$

Were we allowed to use global proof style it would have been easy to establish the global invariant:

$$\Box([\textit{at } \ell_0 \equiv (\textit{at } m_0 \vee \textit{at } m_c)] \wedge [(\textit{at } \ell_1 \vee \textit{at } \ell_c) \equiv \textit{at } m_1])$$

from which $\varphi$ immediately follows.

In the modular style we can establish first

$$\begin{aligned} &\varphi_1 : \Box\{\textit{at } \ell_c \rightarrow (\neg b S(a \wedge \neg b))\} \quad \text{for } S_1, \text{ and} \\ &\varphi_2 : \Box\{\textit{at } m_c \rightarrow (\neg a S(b \wedge \neg a))\} \quad \text{for } S_2 \end{aligned}$$

Here $a$ and $b$ are propositions that hold exactly when the $a$ and $b$ communications take place.

From these we again obtain $\varphi$ immediately. Note that we replaced the forbidden location pointers by a behavioral description of the histories leading to $\ell_c$ and $m_c$ respectively.

Consider the slightly modified program:

$$\begin{aligned} S_3 :: *[[&c![\textbf{true} \rightarrow \ell_c : \text{ Critical Section } \ [\!]\ \textbf{true} \rightarrow \textbf{skip}]]; \\ &[c! \rightarrow \textbf{skip}]] \qquad \| \\ S_4 :: *[[&c? \rightarrow \textbf{skip}]; \\ &[c? \rightarrow [\textbf{true} \rightarrow m_c : \text{ Critical Section } \ [\!]\ \textbf{true} \rightarrow \textbf{skip}]]] \end{aligned}$$

The differences between $S_3 \parallel S_4$ and $S_1 \parallel S_2$ are that the two communication $a$ and $b$ have been identified into a single $c$, and the entry into the critical section has been made non-deterministic.

Using $\mathrm{QTLF_H}$ it is easy to handle this case with similar ease. We take

$$\varphi_1 : \Box\{at\ \ell_c \rightarrow [(true^*; c; true^*; c)^*; true^*; c; true^*]_{\mathrm{H}}\}$$

i.e., $at\ell_c$ may hold only after an *odd* number of $c$'s.

$$\varphi_2 : \Box\{at m_c \rightarrow [(true^*; c; true^*; c)^*; true^*]_{\mathrm{H}}\}$$

In comparison $at\ m_c$ may hold only after an *even* number of $c$'s.

Obviously $\varphi$ follows.

By the previous equivalence results, these formulae can be expressed also in QTL.

Can we do equally well in TL? If we could, then we could count modulo 2 in TL. As already shown in [W] this is impossible. We must conclude:

Proposition 3

Unextended TL is inadequate for modular verification.

In comparison we have:

Proposition 4

QTL is adequate for modular verification.

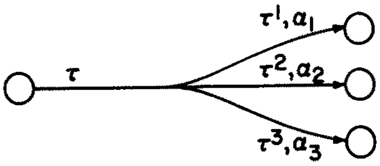
Consider a generic program $S_1 \parallel S_2$. Let $\overline{y}_1$ be all the externally observable variables of $S_1$ and $\overline{x}_1$ the externally unobservables in $S_1$, including the location pointers. It is a standard exercise to define $\psi_1(\overline{x}_1; \overline{y}_1)$ a TL formula that fully specifies all the computations admitted by $S_1$. Similarly construct $\psi_2(\overline{x}_2; \overline{y}_2)$. We now take $\varphi_1 : \exists\overline{x}_1\psi_1(\overline{x}_1; \overline{y}_1)$ and $\varphi_2 : \exists\overline{x}_2\psi_2(\overline{x}_2; \overline{y}_2)$. Note that $\varphi_1$ and $\varphi_2$ obey the modularity principle: they do not have any unobservables as free variables. The proof of $\varphi_1 \wedge \varphi_2 \rightarrow \varphi$ is easy to establish. Since we know that TL is complete for global proofs we may use the complete TL proof system to prove $\psi_1 \wedge \psi_2 \rightarrow \varphi$. We only need the simple $\exists$-introduction rules in order to infer $\varphi_1 \wedge \varphi_2 \rightarrow \varphi$ as required.

## P-Validity and $\alpha$-Fairness

In [Pn3] the problem of verification of concurrent probabilistic programs was studied. It was suggested that the probabilistic behaviour of *finite state* programs can be captured by a special type of fairness, called extreme fairness. It was shown that every temporal property that is proved under the assumption of extreme fairness, is also P-valid, i.e., satisfied with probability 1 over all probabilistic computations of the program. However, we also showed a particular property and program, such that the property was P-valid but could not be proven using extreme fairness.

In this paper we suggest another notion of fairness, called $\alpha$-fairness, that captures precisely the extent of P-validity. That is, every property is P-valid *iff* it holds over all $\alpha$-fair computations.

For our computational model we consider finite state concurrent probabilistic programs that are represented by a finite graph with labelled complex transitions. Each node in the graph corresponds to a global state of the program (i.e., in case the program was originally represented as several separate processes, the graph we consider here is their cartesian product). A transition



in the graph is a complex edge which has a single source and several destinations. The branching structure represents the probabilistic multiplicity of outcomes in which the execution of $\tau$ may result. The different branches $\tau', \ldots \tau^m$ are called the ***modes*** of $\tau$ and they are each associated with a positive probabilities $\alpha_i > 0$, $i = 1, \ldots m$. The transitions $\tau$ are partitioned into processes $P_1, \ldots, P_k$ such that the set of transitions associated with $P_j$ are all the transitions that $P_j$ may effect. We require that for each node $n$ and each process $P_j$, $j = 1, \ldots k$ there is a transition belonging to $P_j$ that departs from $n$. One node in the graph is distinguished at the ***initial node.*** It corresponds to the initial global state.

A ***computation tree*** for a program $G$ is a infinite unwinding tree of $G$ that satisfies the following:

a. The root of the computation tree is the initial node.

b. For each instance of a node $n$ appearing in the tree we choose one transition $\tau$ that departs from $n$ in $G$, and include in the tree all of its modes and instances of all the nodes that are its destinations in $G$.

c. (Justice) Each (infinite) path in the tree must contain infinitely many $P_j$-transitions for each $P_j$, $j = 1, \ldots k$.

For example, the tree of Figure 2 is a computation tree for the program in Fig. 1.

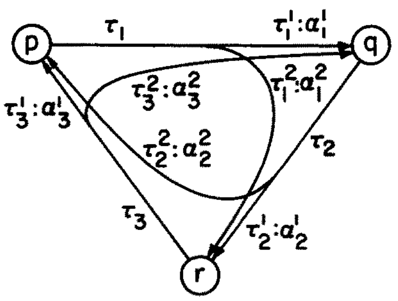


Figure 1

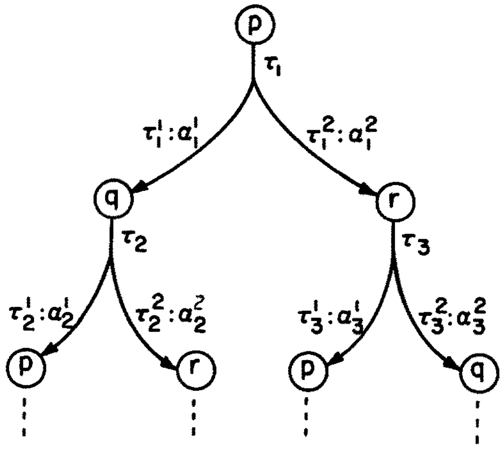


figure 2

The probabilities associated with the modes in the tree induce a natural probability measure on the paths in $T$.

Thus, for example the measure of the set of all paths that share a common prefix of the form:

$$\begin{tikzcd} n_0 \arrow[r, "\alpha_1"] & n_1 \arrow[r, "\alpha_2"] & n_2 \cdots \arrow[r, "\alpha_r"] & n_r \end{tikzcd}$$

is $\alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_n$.

Actually, we may relax requirement c. above by requiring that the set of just paths in $T$ is of measure 1.

A path property $\psi$ is said to hold with probability 1 over a computation tree $T$, if the set of paths in $T$ that satisfy $\psi$ has a measure 1. A path property $\psi$ is said to be ***P-valid*** (probabilistic valid) for a program $G$, if it holds with probability 1 over all computation trees of $G$.

A ***computation*** for a program $G$ is an infinite path in some computation tree. Note that a computation singles out a particular mode $\tau^j$ for each transition $\tau$.

Let $\tau$ be a probabilistic transition that can result in *m modes*, $\tau^1, \ldots \tau^m$. Let $\beta$ be a star-free (regular) expression and $\sigma$ an infinite computation. A finite prefix $\sigma' < \sigma$ is said to be a ***$\beta$-prefix*** if $\sigma' \models \beta$. The computation $\sigma$ is said to be ***$\gamma$-fair*** (***$\alpha$-fair***) with respect to the transition $\tau$ and the star-free (regular) expression $\beta$, if *either* the $\tau$ transition is taken only finitely many times from $\beta$-prefices, *or* each of the $\tau^1, \ldots, \tau^m$ modes is taken infinitely many times from $\beta$-prefices in $\sigma$.

The computation $\sigma$ is defined to be ***$\gamma$-fair*** (***$\alpha$-fair***) if it is $\gamma$-fair ($\alpha$-fair) with respect to all the probabilistic transitions in the program, and all the star-free (regular) expressions $\beta$.

A TL (QTL) formula $\psi$ is said to be ***$\gamma$-valid*** (***$\alpha$-valid***) over a program $G$, if it is valid over all the $\gamma$-fair ($\alpha$-fair) computations of $G$.

**Lemma 1 (Completeness)**

Let $\psi$ be a TL (QTL) formula.

$$\psi \text{ is not } \gamma\text{-valid } (\alpha\text{-valid}) \implies \psi \text{ is not P-valid.}$$

Outline of Proof

To establish the lemma, we assume a computation $w$ that is $\gamma$-fair ($\alpha$-fair) and does not satisfy $\psi$. As $\neg\psi$ is a TL (QTL) formula, there exists an $\omega$-counter-free ($\omega$-regular) automaton that defines $\neg\psi$. We consider one of its accepting runs on $w$. This run can be viewed as a sequence of pairs, where each pair consists of a program state and an automaton state. Let $\hat{w}$ denote the suffix of the sequence that satisfies:

a. Each pair appearing at least once in $\hat{w}$ appears there infinitely many times.

b. Each combination of consecutive pairs that appears in $\hat{w}$ at least once, appears there infinitely many times.

The number of distinct pairs in $\hat{w}$ is finite. Assume there are $m$ such pairs, $\Pi_1, \ldots, \Pi_m$. For each such pair $\Pi_i$ $(1 \leq i \leq m)$, we assign a set of processes

$$h_i = \{P_1^i, \ldots, P_{n_i}^i\}$$

which is the set of processes activated from $\Pi_i$ in $\hat{w}$. As $w$ is $\gamma$-fair ($\alpha$-fair), it is also just, and

$$\bigcup_{i=1}^{m} h_i = \mathcal{P} \qquad \text{where } \mathcal{P} = \{P_1, \ldots, P_k\} \text{ is the set of all processes.}$$

The sequence $\hat{w}$ can be represented by

$$\hat{w} = \Pi^0, \Pi^1, \ldots \qquad (\Pi^j \in \{\Pi_1, \ldots, \Pi_m\})$$

From $\hat{w}$ we construct a computation tree $T$ as follows:
Each vertex in the tree is labelled by some

$$\Pi_i \times |h_1| \times \ldots \times |h_m| \qquad i \in \{1 \ldots m\}$$

The root of the tree is labelled by

$$(\Pi^0, 1, \ldots, 1)$$

The tree is inductively constructed as follows:

From a leaf $v$ labelled by $(\Pi_j, i_1, \ldots, i_m)$, the process $P_k = P^j_{i_j}$ is activated. For each $\Pi_l$ such that $\Pi_j \xrightarrow{P_k} \Pi_l$, a vertex labelled by

$$(\Pi_l, i_1, \ldots, (i_j \bmod n_j) + 1, \ldots, i_m)$$

is constructed, and connected as a son of $v$.

We claim that the computation tree is fair with probability 1, and satisfies $\neg\psi$ with probability 1. This is proved by showing the existence of a finite path of bounded length, from each vertex $(\Pi_j, \ldots)$ to each vertex $(\Pi_k, \ldots)$, for each $1 \leq j, k \leq m$. From that, we can derive that an infinite path in $T$ reaches each vertex of the type $(\Pi_k, \ldots)$ $(1 \leq k \leq m)$ infinitely many times, with probability 1. Each such path has an infinity set of states that is identical to the infinity set of states of the $\omega$-counter-free ($\omega$-regular) automaton that defines $\psi$, when running on $w$, and thus satisfies $\neg\psi$. Hence, the tree satisfies $\neg\psi$ with probability 1.

In a path that reaches every vertex of the type $(\Pi_j, \ldots)$ $(1 \leq j \leq m)$ infinitely many times, each process of $h_j$ is taken infinitely many times. As $\bigcup_{i=1}^{m} h_i = \mathcal{P}$, it follows that the tree is just with probability 1.

Given a TL (QTL) formula $\psi$, we constructed a fair computation tree in which $\neg\psi$ is satisfied with probability 1. It follows that $\psi$ is not P-valid. ▮

**Lemma 2 (Soundness)**

Let $\psi$ be a TL (QTL) formula.

$$\psi \text{ is } \gamma\text{-valid } (\alpha\text{-valid}) \Longrightarrow \psi \text{ is P-valid.}$$

Outline of Proof

The statement that $\psi$ is $\gamma$-valid ($\alpha$-valid) means that all $\gamma$-fair ($\alpha$-fair) computations satisfy $\psi$. We proceed to show that the measure of computations which are not $\gamma$-fair ($\alpha$-fair) is 0, leaving a measure 1 set of computations that satisfy $\psi$.

Let $G$ be a program with a bounded probability distribution, and $T$ one of its computation trees. Define $COM_\beta$ to be the set of all computations of $T$, which are not $\gamma$-fair ($\alpha$-fair) with respect to a star-free (regular) expression $\beta$. Thus, $\sigma \in COM_\beta$ if in $\sigma$ some probabilistic transition $\tau$ is taken infinitely often after $\beta$-prefices, and some mode $\tau^j$ of $\tau$ is taken only finitely often after $\beta$-prefices.

The set $COM_\beta$ can be partitioned to $COM_\beta^{\tau^j}$, according to the neglected mode $\tau^j$. (The $COM_\beta^{\tau^j}$-s may not be disjoint).

We claim that for each $\tau^j$, $\mu(COM_\beta^{\tau^j}) = 0$.

Taking the union of $COM_\beta^{\tau^j}$ over the finitely many different $\tau^j$-s in the program $G$ and the countable star-free (regular) expressions on the alphabet $\Sigma = \{\text{The set of states in } G\}$, we obtain a set of measure 0. We conclude that the set of computations in $T$ which are not $\gamma$-fair ($\alpha$-fair) is of measure 0. ▮

## Discussions

A version of TL which does not assume bounded past is also possible. The approach we have taken here can be easily extended to cover this version. We have to consider sequences that may be infinite at both directions (i.e., indexed by positive and negative integers). In the decision procedure we have to define a criterion of self fulfillment of initial MSCS's that requires the presence of the formula $\psi_2$ in the component for each formula $\psi_1 \mathcal{S} \psi_2$ that appears in the component. In the axiomatic system we have to replace the clause $\diamondminus\circleddash \mathit{false}$ in $A$ by $p\,\mathcal{S}q \rightarrow \diamondminus q$.

Another extension which is easy to obtain is to consider formulae over a given finite state program. This is usually referred to as model checking. The decision procedure is very similar to the one presented in [LP]. For a corresponding deductive system it is sufficient to add the program dependent axioms of [LP] in order to ensure a complete system.

## Acknowledgement

We wish to thank Joe Halpern for helpful suggestions, Carol Weintraub for her excellent typing and Y. Barbut for the drawings.

## References

[AS] Alpern, B., Schneider, F.B., - Defining Liveness, Cornell University, (Oct. 1984).

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

[BKP] Barringer, H., Kuiper, R., Pnueli, A. - Now You May Compose Temporal Logic Specifications, *16th Symposium on Theory of Computing* (April 84), 51-63.

[Buc] Büchi, J.R., - On a Decision Method in Restricted Second Order Arithmetic, *Proc. Intern. Congr. Logic, and Philos. Sci. 1960*, 1960, Stanford University Press (1962) 1-11.

[Buc] Büchi, J.R., - Weak Second Order Arithmetics and Finite Automata, Z. Math. Logik Grundlagen Math 6 (1960) 66-92.

[Bur] Burgess, J., - Basic Tense Logic, in Handbook of Philosophical Logic, Vol. 2 (D. Gabbay and F. Guenthner eds.) D. Reidel Pub. Co., (1984).

[C] Choueka, Y., - Theories of Automata on $\omega$-Tapes: A Simplified Approach, *Journal of Computers and Systems Sciences* 8 (1974) 117-141.

[CH] Chen, Z.C., Hoare, C.A.R., - Partial Correctness of Communicating Processes and Protocols, Technical Monograph, PRG-20 Oxford University Computing Laboratory (May 1981).

[GPSS] Gabbay, D., Pnueli, A., Shelah, S., Stavi, J., - On the Temporal Analysis of Fairness, *Proc of the 7th ACM Symp. on Principles of Programming Languages* (1980) 163-173.

[H] Hoare, C.A.R., - A Calculus of Total Correctness for Communicating Precesses, Technical Monograph, RPG-23 Oxford University Computing Laboratory (May 1981).

[HO] Hailpern, B., Owicki, S., - Modular Verification of Computer Communication Protocols, *IEEE Trans. on Communications*, COM-31, 1 (Jan. 1983) 56-68.

[K] Kamp, H.W., - Tense Logic and the Theory of Linear Order, Ph.D. Thesis, University of California Los Angeles (1968).

[KVR] Koymans, R., Vytopil, J., DeRoever, W.P., - Real Time Programming and Asynchronous Message Passing, *2nd ACM Symp. of Distributed Computing*, Montreal (1983) 187-197.

[L] Lamport, L., - What is Good in Temporal Logic?, Proceeding IFIP (1983) 657-668.

[Li] Lichtenstein, O., - Decidablity and Completeness of a Temporal Proof System for Finite State Programs, M.Sc. Thesis, Tel Aviv University (1984).

[LP] Lichtenstein, O., Pnueli, A., - Checking That Finite State Concurrent Programs Satisfy Their Linear Specification, *ACM Symp. on Principles of Programming Languages* (1985).

[MC] Misra, J., Chandy, K.M., - Proofs of Networks of Processes, *IEEE Trans. on Software Engineering* 5E-7, 4 (July 1981).

[MNP] McNaughton, R., Papert, S., - Counter Free Automata, MIT press, Cambridge, Mass (1971).

[MP1] Manna, Z., Pnueli, A., - Verification of Concurrent Programs: A Temporal Proof System, *Proc. 4th School on Advanced Programming*, Amsterdam (June 1982) 163-255.

[MP2] Manna, Z., Pnueli, A., - How to Cook a Temporal Proof System for your Pet Programming Language *Proc of the 10th ACM Symp. on Principles of Programming Languages* (1983).

[MP3] Manna, Z., Pnueli, A., - Adequate Proof Principles for Invariance and Liveness Properties of Concurrent Programs, Science and Computer Programming, Forthcoming.

[NGO] Nguyen, V., Gries, D., Owicki, S., - A Model and Temporal Proof System for Networks of Processes, *Proc of the 12th ACM Symp. on Principles of Programming Languages*

(1985).

[OG] Owicki, S., Gries, D., - An axiomatic Proof Technique for Parallel Programs, *Acta Informatica 6* (1976) 319-340.

[OL] Owicki, S., Lamport, L., - Proving Liveness Properties of Concurrent Programs, *ACM TOPLAS 4,3* (July 1982) 455-495.

[Pe] Peikert, R., - Propositional Temporal Logic and $\omega$-regular Languages, Manuscript, ETH-Zürich.

[Pn1] Pnueli, A., - The Temporal Logic of Programs, *18th Annual Symp. on Foundation of Computer Science* (1977) 46-57.

[Pn2] Pnueli, A., - In Transition from Global to Modular Temporal Reasoning about Programs, *Proc. Advanced NATO Institute on Logic and Models for Verification and Specification of Concurrent Systems*, La Colle-Sur-Loupe (Oct. 1984).

[Pn3] Pnueli, A., - On the Extremely Fair Treatment of Probabilistic Algorithms, *Proc. of the 15th Annual ACM Symp. on Theory of Computing* (1983).

[Pr] Prior, - Past Present and Future, Oxford Press.

[RU] Rescher, N., Urquhart, A., - Temporal Logic, Springer Verlag (1971).

[S] Sistla, A.P., - Characterization of Safety and Liveness Properties in Temporal Logic, University of Massachusetts, Amherst (Nov. 1984).

[SC] A.P. Sistla, E.M. Carke, The Complexity of Propositional Temporal Logic, *14th ACM Symposium on Thoery of Computing*, (May 1982) 159–167.

[T] Thomas, W., - A Combinatorial Approach to the Theory of $\omega$-Automata, *Information and Control 48,3* (March 1981) 261-283.

[VW] Vardi, M.Y., Wolper, P., - Expressiveness and Complexity of Languages for Describing Sequences, Stanford University.

[W] Wolper, P., - Temporal Logic can be More Expressive, *Proc. of the 22nd Annual Symp. on Foundation of Computer Science* (1981) 340-348.