Safety, Liveness and Fairness in Temporal Logic*

A. Prasad Sistla
Department of Electrical Engineering and Computer Science
University of Illinois at Chicago

August 6, 1999

Abstract

In this paper we present syntactic characterization of temporal formulas that express various properties of interest in the verification of concurrent programs. Such a characterization helps us in choosing the right techniques for proving correctness with respect to these properties. The properties that we consider include safety properties, liveness properties and fairness properties. We also present algorithms for checking if a given temporal formula expresses any of these properties.

Key Words: Concurrent Programs, Verification, Temporal Logic, Safety Properties, Liveness Properties, Fairness Properties, Algorithms.

*This work is partly supported by NSF grant CCR-9212183. A preliminary version of this paper appeared in the Fourth ACM Symposium on Principles of Distributed Computing.

1 Introduction

In the verification of concurrent programs two kinds of properties are of primary importance and have been extensively investigated ([La77]): *safety properties* and *liveness properties*. Safety properties assert that something bad never happens, while liveness properties assert that something good will eventually happen. The classification of properties into safety properties and liveness properties allows us to choose the most appropriate proof method for proving correctness with respect to these properties. For example, methods based on global invariants have been extensively used for safety properties, while methods based on proof lattices or well-founded induction have been employed for liveness properties (see [OL82]).

Temporal Logic has been proposed in [Pn77] (also see [MP92]) as an appropriate formalism in the specification and verification of concurrent programs. Since then, many different versions of temporal logics have been used in the verification of concurrent programs [MP89, LPZ85, SC85, CES86]. Due to this wide interest, it becomes important to present a syntactic classification of formulas in temporal logic that specify safety and liveness properties.

Knowing whether a formula specifies a safety or a liveness property, helps us in choosing the right proof method for verifying that a given concurrent program satisfies the property specified by the formula. Additionally, it is well known that all *fair* executions [1] of a program satisfy a safety property iff all executions (fair or unfair) satisfy the property. This means that we need not be concerned about fairness for proving safety properties. On the other hand, fairness is often essential in proving general liveness properties. Thirdly, proving that a program satisfies a property may not always be feasible due to many reasons. In such cases, monitoring the executions of a program for violations of the property may be an alternative. It is possible, in principle, to monitor an execution of a concurrent program for violations of safety properties. In contrast, this is not possible for liveness properties. Thus, knowning if a formula expresses a safety property helps us determine whether we can monitor the execution of a program for violations of the property. Motivated by such concerns, in this paper, we investigate the possibility of syntactically characterizing safety, liveness and fairness properties in Propositional Linear Temporal Logic(PTL).

Formally, a property is simply a set of infinite sequences of states. A definition of safety properties was first given by Lamport [La85] (we call these as L-safety properties), and a more general definition is given in [AS85]. We focus on the later definition. A safety property is same as a limit closure property defined in [Em83]. In [ADS86], it has been shown that L-safety properties is same as the class of safety properties that are closed under stuttering. Intuitively, closure under stuttering requires that the property be insensitive to successive repetition of any state of a sequence.

In this paper, we introduce a new class of properties called *strong safety* properties. A property C is a strong safety safety property if it is a safety property that is closed under stuttering, and is insensitive to deletion of states, i.e. from any sequence in C if we delete an arbitrary number of states, then the resulting sequence is also in C. The class of strong safety properties is a strict subset of the class of safety properties that are closed under stuttering.

We give a syntactic characterization of PTL formulas that express safety properties,

[1] We give a formal definition of fairness properties, which may be viewed as a special class of liveness properties.

safety properties closed under stuttering and strong safety properties. Specifically, we show that all positive formulas formed using $\mathcal{U}$ (*unless*) and $\bigcirc$ (*nexttime*) express safety properties, and all positive formulas formed using only $\mathcal{U}$ express safety properties that are also closed under stuttering. We also show that all positive formulas formed using $\Box$ (*always*) express strong safety properties. In this case, we show the completeness result also, i.e. every strong safety property expressible in PTL can be expressed using positive formulas that only use the $\Box$ modal operator. In fact, we prove a stronger result that shows that any strong safety property expressed by a finite state automaton can be expressed by a positive formula that only uses the $\Box$ temporal operator.

We also consider *liveness* properties. Here again we use the definition of [AS85]. Intuitively, a property is a liveness property if every finite sequence can be extended to an infinite sequence that satisfies the property. We introduce a new class of properties called *absolute liveness* properties. A property is an absolute liveness property if it is a liveness property, and is insensitive to addition of prefixes, i.e. for any sequence in the property if we add an arbitrary prefix to it then the resulting sequence is also in the property. We give a simple characterization of PTL formulae that express absolute liveness property. We also give a formal definition of *stable* properties and *fairness* properties. A stable property is a property that is insensitive to deletion of prefixes, i.e. it is closed under suffixes. A fairness property is a property that is insensitive to addition or deletion of prefixes. Fairness properties play an important role in verifying liveness properties of concurrent programs. We characterize the formulae that express both of the above types of properties.

In addition to presenting syntactic characterizations, we also present algorithms for recognizing formulas that express safety properties and liveness properties. Finally, we show that the set of formulae that express safety properties is PSPACE-complete. We present a double exponential time algorithm for recognizing formulas that express liveness properties.

The paper is organized as follows. Section 2 contains the definitions of the temporal logic that we use, and the definitions of the different properties that we consider in the paper. Section 3 presents the results on safety and strong safety properties. Section 4 considers characterization of absolute liveness and other properties. Section 5, gives the decision procedures for detecting if a given formula expresses a safety property or a liveness property. Finally, section 6 contains the concluding remarks.

2 Notation and Definitions

2.1 Linear Temporal Logic

The language of Propositional Linear Temporal Logic(PTL) uses the modalities $\bigcirc$(*nexttime*), $\mathcal{U}$ (*unless*) together with symbols for atomic propositions drawn from a finite set F, the propositional connectives $\wedge, \neg$ and parenthesis. The set of well formed formulae in PTL is the smallest set satisfying the following conditions: every atomic proposition from F is a well formed formula; *true* and *false* are well formed formulae; if f and g are well formed formulae then so are $f \wedge g$, $\neg f$, $\bigcirc f$ and $f\ \mathcal{U}\ g$.

A *state* is a mapping from F into the set $\{\textbf{true}, \textbf{false}\}$. We let Σ denote the set of all states. Since F is a finite set, Σ is also a finite set. An interpretation is a pair (s, i) where $s = s_0, s_1, ..., s_i, ...$ is a ω-sequence of states and i is a non-negative integer. We define a satisfiability relation, $\models$, between an interpretation and a PTL formula f inductively on the structure of f as follows.

- $s, i \models \mathit{true}$ and $s, i \not\models \mathit{false}$ for every $s \in \Sigma^\omega$ and $i \geq 0$,
- $s, i \models P$ for $P \in \mathsf{F}$ *iff* $s_0(P) = \mathbf{true}$,
- $s, i \models \neg f$ iff $s, i \not\models f$,
- $s, i \models f \wedge g$ iff $s, i \models f$ and $s, i \models g$,
- $s, i \models f\ \mathcal{U}\ g$ iff either for all $k \geq i$ $s, k \models f$, or there exists some $j \geq i$, $s, j \models g$ and for every k, $i \leq k < j$, $s, k \models f$.
- $s, i \models \bigcirc f$ iff $s, i+1 \models f$.

Note that $f\ \mathcal{U}\ g$ does not imply that g is eventually satisfied. The operator $\mathcal{U}$ is also called *weak until* in the literature. We say that an infinite sequence of states s *satisfies* a formula f if $s, 0 \models f$. We say that a PTL formula f is *satisfiable* if there exists some $s \in \Sigma^\omega$ that satisfies f.

In addition to the above temporal operators, we also use the derived unary operators $\Box$ and $\Diamond$ and the derived propositional connectives $\vee$, $\rightarrow$ and $\leftrightarrow$ defined as follows: $\Box f \equiv f\ \mathcal{U}\ \mathit{false}$, $\Diamond f \equiv \neg\Box\neg f$, $f \vee g \equiv \neg(\neg f \wedge \neg g)$, $f \rightarrow g \equiv \neg f \vee g$, $f \leftrightarrow g \equiv (f \rightarrow g) \wedge (g \rightarrow f)$. Notice that $s, i \models \Box f$ iff for all $j \geq i$, $s, j \models f$, and $s, i \models \Diamond f$ iff for some $j \geq i$ $s, j \models f$. Although, we have defined $\Box$ and $\Diamond$ in terms of the $\mathcal{U}$ operator, we treat them as separate operators.

For a set M of temporal modal operators, the set of positive formulae using these operators are exactly those that are formed using the modalities from M and the propositional connectives $\wedge, \vee, \neg$ with none of the modalities of M appearing in the scope of $\neg$.

2.2 Safety Properties and Liveness Properties

Now, we give the definitions of different properties that we intend to characterize. We use the following notation throughout the paper. For any set Δ, Δ^* and Δ^ω denote the set of all finite sequences and the set of all infinite sequences, i.e. ω-sequences, over Δ. For any ω-sequence $t = (t_0, t_1, ...) \in \Sigma^\omega$ and non-negative integers i and j such that $j \geq i$, we define the sequences $t(i)$ and $t(i, j)$ as follows: $t(i) = (t_0, t_1, ..., t_i)$ and $t(i, j) = (t_i, t_{i+1}, ..., t_j)$. Note that $t(i)$ is the prefix of t consisting of the first $i + 1$ elements and $t(i, j)$ is the sequence of elements in t between the $i + 1$st and $j + 1$st elements. We define $t(i, \infty)$ to be the ω-sequence $t_i, t_{i+1}, ...$; this is the suffix of t starting from the $i + 1$st state of t. A *property* is a subset of Σ^ω. For a PTL formula f the property expressed by f is the set $\{t \in \Sigma^\omega : (t, 0) \models f\}$.

Safety Properties

The following definition of safety property is given in [AS86]. A property C is a safety property iff the following condition is satisfied for all $t \in \Sigma^\omega$:

if $\forall i \geq 0, \exists u \in \Sigma^\omega$ such that $t(i)u \in C$ then $t \in C$.

Note that the limit closure properties of [Em83] are exactly the safety properties. According to this definition, if C is a safety property and t is an ω-sequence each of whose prefixes can be extended to an ω-sequence in C then t is also in C. Stated differently, if C is a safety property and t is not in C then for some prefix of t none of its extensions to an ω-sequence is in C.

A property C is said to be closed under *stuttering* if $t = t_0, t_1, .., t_i, t_{i+1}, ...$ is in C then the sequence $t_0, t_1, ..., t_i, t_i, t_{i+1}, ...$ is also in C. If C is closed under stuttering and $t \in C$ then if we successively repeat any state of t an arbitrary finite number of times then the resulting sequence will also be in C.

Lamport had proposed a different definition of safety properties. We will refer to these properties as L-safety properties. A property C is a L-safety property iff the following condition is satisfied:

$$\forall t = (t_0, ..., t_i, ...) \in \Sigma^\omega,\ t \in C \text{ iff } \forall i \geq 0, t(i)t_i^\omega \in C.$$

Let us say that an extended prefix of a sequence of states t is an ω-sequence of states obtained by taking a prefix of t and extending it by repeating its last state infinitely many times. If C is a L-safety property and t is any ω-sequence of states all of whose extended prefixes are in C then t is also in C. It has been shown in [AS86] that a property which is closed under stuttering is a L-safety property iff it is a safety property. We will refer to such properties as *safety properties with stuttering.* A property C is a *strong safety property* iff

- (a) C is a safety property with stuttering and
- (b) $\forall t = t_0, t_1... \in C, \forall i > 0$ the sequence $t_0, t_1, ..., t_{i-1}, t_{i+1}, ... \in C$, i.e., if $t \in C$ and we delete any state other than the initial state from t then the resulting sequence should also be in C.

The motivation for condition (b) in the above definition is that if we don't observe the system at certain times then the observed behavior should still satisfy the property(it is assumed that the initial state is always observed).

Invariant properties are properties that are expressed by the formulae of the form $\Box f$ where f is a propositional formula. Clearly, the class of invariant properties is a subclass of strong safety properties.

A formula is a *safety formula* iff it expresses a safety property and is a *strong safety formula* iff it expresses a strong safety property.

Liveness Properties

Now, we give the definition of a liveness property. A property C is a *liveness property* iff the set $\{t(i) : t \in C\}$ is Σ^*. A formula is a *liveness formula* iff it expresses a liveness property. Intuitively, a formula is a liveness formula iff every finite sequence can be extended to satisfy the formula. A property C is an *absolute liveness* property iff C is non-empty and $\forall t \in C, \forall u \in \Sigma^*, ut \in C$. Note that every absolute liveness property is also a liveness property.

A property C is a *stable* property if C is non-empty and for every sequence in C all it's suffixes are also in C, i.e. C is closed under suffixes. If f expresses a stable property and if $t, 0 \models f$ then $\forall i \geq 0, t, i \models f$.

LEMMA 2.1: For a non-empty property C, C is a stable property iff the complement of C is an absolute liveness property.

Proof: Let $D = \Sigma^\omega - C$. Assume that C is a stable property. We like to show that D is an absolute liveness property. Now, let $t \in D$ and u be any finite sequence in Σ^*. If $ut \in C$ then it would imply that $t \in C$ (because C is closed under suffixes) which contradicts the assumption that $t \in D$. Hence, $\forall u \in \Sigma^*, ut \in D$. Hence D is an absolute liveness property.

Now assume that D is an absolute liveness property. We like to show that C is a stable property. Let $t \in C$ and t' be any suffix of t. If $t' \in D$ then from the definition of an absolute liveness property it would immediately follow that $t \in D$, which is a contradiction. Hence $t' \in C$ and C is closed under suffixes and is a stable property. ■

Fairness properties form another important class of properties. Usually a fairness property only talks about the infinite behavior of computations. For this reason we use the following definition. A property C is a fairness property iff C is a stable property and is an absolute liveness property. See [Fr86] for a discussion of various fairness properties. Also [EL87] considers various fairness properties expressed in temporal logic. They do not, however, give a semantic definition of fairness properties.

Examples

Now, we give some simple examples of different safety and liveness properties that we defined above. Let $a, b \in \Sigma$. Then $ab\Sigma^\omega$, which is the set of sequences starting with a immediately followed by b, is a safety property. It is not a safety property with stuttering. This is due to the fact that if we repeat the first state twice then the resulting sequence is not in C. The set $a^+b^+\Sigma^\omega$ is a safety property with stuttering, but is not a strong safety property. Here a^+ is one or more occurrences of a. The set $a^*b^*\Sigma^\omega$ is a strong safety property. Here a^* denotes zero or more occurrences of a.

The set $(\Sigma - \{a\})^\omega \cup (\Sigma - \{a\})^* a \Sigma^* b \Sigma^\omega$ is a liveness property which is not an absolute liveness property. It is the set of sequences that have an occurrence of b following some time after the first occurrence of a. The set $\Sigma^* a \Sigma^\omega$, which is the set of all strings that contain at least one a is an absolute liveness property.

Below, we present some practical examples.

The simple safety property for resources is that the resource is never allocated unless there is a request for it. Let *request*,*alloc* be atomic propositions indicating requesting of the resource and allocation of the resource, respectively. Let REQ denote the set of states that assign value *true* to the proposition *request*. Similarly, let ALLOC denote the set of states that assign value *true* to the proposition *alloc*. Formally, the simple safety property for resources, denoted by *safe* , is the set of sequences in which every occurrence of an ALLOC state is preceded by a REQ state. This property is expressed by the following formula:

$$\neg alloc \ \ \mathcal{U} \ \ request$$

The safety property for continuous resource allocation, denoted by *con-safe* , is the set of sequences in which every occurrence of an ALLOC state is precede by a REQ state, and in addition between every two ALLOC states there is a REQ state. This is expressed by the following formula:

$$(\neg alloc \ \ \mathcal{U} \ \ request) \wedge \Box(alloc \rightarrow \bigcirc[\neg alloc \ \ \mathcal{U} \ \ request])$$

The bounded safety property for resources, denoted by *bounded-safe* , is the set of sequences in which, whenever a state from REQ occurs then with in the next two states a state from ALLOC also occurs. This is expressed by the following formula:

$$\Box(\mathit{request} \rightarrow (\mathit{alloc} \vee \bigcirc\ \mathit{alloc} \vee \bigcirc\bigcirc\ \mathit{alloc}))$$

From the definition of safety properties, it can be shown that both *safe* , *con-safe* and *bounded-safe* are all safety properties. In addition, *safe* is closed under stuttering while *con-safe* and *bounded-safe* are not. Thus, *safe* is also a safety property with stuttering.

Let *monotone-p* be the set of sequences t satisfying the following condition: either P is satisfied in all states of t, or if P is not satisfied in any state of t then P is not satisfied in all future states from that point onwards. *monotone-p* is a strong safety property. This is expressed by the following formula:

$$\Box(P \vee \Box\neg P)$$

A simple invariant property is the absence of dead locks. Let *no-deadlock* be an atomic proposition that indicates the absence of dead locks in a state. The formula $\Box$*no-deadlock* asserts dead lock never in a computation.

Now, we present some examples of liveness properties. Let *live-resource* be the set of sequences in which every occurrence of a REQ state is eventually followed by an ALLOC state. *live-resource* is a liveness property. It is not an absolute liveness property due to the following reason. Consider a sequence t that does not contain any REQ or ALLOC states. This sequence is in *live-resource* . However the sequence rt, where r is a state in REQ , is not in *live-resource* . *live-resource* is expressed by the following formula.

$$\Box(\mathit{request} \rightarrow \Diamond\ \mathit{alloc})$$

Let *terminate* be an atomic proposition that indicates the termination of a program. Also, let TERM be the set of states in which *terminate* is assigned the value *true*. The set of all sequences of states that contain a state from TERM is an absolute liveness property. This property is expressed by the formula $\Diamond$*terminate*.

An example of a stable property is an invariance property. Now we give an example of a fairness property. Let *send* and *receive* be atomic propositions that denote sending and receiving of any message on a channel. Let SEND be the set of states that assign the truth value *true* to the proposition *send*, and RECEIVE be the set of states that assign *true* to the proposition *receive*. Let *channel-fairness* be the set of all sequences t satisfying the following property: if t contains an infinite number of occurrences of SEND states then it also contains an infinite number of occurrences of RECEIVE states. *channel-fairness* is a fairness property. It is expressed by the following formula:

$$\Box\Diamond\ \mathit{send}\ \rightarrow\ \Box\Diamond\ \mathit{receive}$$

3 Characterization of Safety Properties

In this section we consider syntactic characterization of PTL formulas that express different safety properties. The results are summarized in table 1 at the end of the section.

3.1 Safety Properties in PTL

The following theorem shows that all positive formulas express safety properties.

THEOREM 3.1: Every propositional formula is a safety formula and if f, g are safety formulae, then so are $f \wedge g, f \vee g, \bigcirc f, f \;\mathcal{U}\; g$ and $\Box f$.

Proof: It is easily seen that every propositional formula expresses a safety property. Assume that f and g express safety properties. It is straightforward to see that $f \wedge g$ expresses a safety property. Now consider $f \vee g$. Let $t \in \Sigma^\omega$ be such that every prefix t' of t can be extended to satisfy $f \vee g$. It is easily seen that either infinitely many prefixes of t can be extended to satisfy f and hence all prefixes of t can be extended to satisfy f, or all prefixes of t can be extended to satisfy g. From this observation and the the assumption that f and g are safety formulae, it follows that either $t, 0 \models f$ or $t, 0 \models g$. Hence $t, 0 \models f \vee g$. From this, it follows that $f \vee g$ is a safety formula. It is easy to see that $\bigcirc f$ expresses a safety property. Now we like to show that $f \;\mathcal{U}\; g$ expresses a safety property. Let $t \in \Sigma^\omega$ be such that $\forall i \geq 0, \exists u \in \Sigma^\omega$ such that $t(i)u \models f \;\mathcal{U}\; g$. Now we have the following two cases:

- Case 1: $\forall i, \forall j$ such that $0 \leq i \leq j < \infty \exists u \in \Sigma^\omega$ such that $t(i,j)u, 0 \models f$. Since f expresses a safety property it follows that $\forall i \geq 0, t(i, \infty), 0 \models f$. Recall that in our notation $t(i, \infty)$ denotes the suffix of t starting from the state t_i. Thus $\forall i \geq 0, t, i \models f$. Hence $t, 0 \models f \;\mathcal{U}\; g$.

- Case 2: Assume that case 1 does not hold. That is, $\exists i, j$ such that $0 \leq i \leq j < \infty$ and $\forall u \in \Sigma^\omega, t(i,j)u, 0 \models \neg f$. Let k be the smallest value of i for which the above condition holds and k' be the corresponding value of j. We know that $\forall i \geq 0 \exists u \in \Sigma^\omega$ such that $t(i)u, 0 \models f \;\mathcal{U}\; g$. Now, consider any $i \geq k'$ and some $u \in \Sigma^\omega$ such that $t(i)u, 0 \models f \;\mathcal{U}\; g$. From our assumption it is the case that $t(k, k')u, 0 \not\models f$ and hence $t(i)u, k \not\models f$. As a consequence, we see that $\exists m \leq k$ such that $t(i)u, m \models g$, and $\forall p$ such that $0 \leq p < m \; t(i)u, p \models f$. This implies that $t(m, i)u, 0 \models g$ and $\forall p$ such that $0 \leq p < m \; t(p, i)u, 0 \models f$. From this, it follows that $\exists m \leq k$ such that for infinitely many values of $n \geq m$, $\exists u \in \Sigma^\omega$, such that $t(m, n)u, 0 \models g$ and $\forall p$ such that $p < m$ $t(p, n)u, 0 \models f$. From this and the assumption that f and g are safety formulae, it easily follows that $t(m, \infty), 0 \models g$ and $\forall p < m \; t(p, \infty), 0 \models f$. Hence $t, m \models g$ and $\forall p < m, t, p \models f$. Thus it is seen that $t, 0 \models f \;\mathcal{U}\; g$.

From the above argument, it clearly follows that $f \;\mathcal{U}\; g$ expresses a safety property. $\Box f$ expresses a safety property since $\Box f \leftrightarrow f \;\mathcal{U}\;$ **false**. ■

Note that all the formulas expressing the different safety properties *safe* , *con-safe* and *bounded-safe* , given in section 2.2 are positive.

It is the case that any PTL formula that is expressively equivalent to a positive formula is also a safety specification. Consider any complete axiomatization for PTL. Let $\vdash f$ indicate that f is theorem in this deductive system. Let SAF be the set of PTL formulae generated by the following rules. It is easily seen that all members of SAF are safety formulae.

- If f is a positive formula then $f \in SAF$;

- If $\vdash (f \leftrightarrow g)$ and $g \in SAF$ then $f \in SAF$.

3.2 Safety with Stuttering

The following theorem shows that all positive formulas formed using $\mathcal{U}$ express safety properties with stuttering.

THEOREM 3.2: Every positive formula using only the operator $\mathcal{U}$ expresses a safety property with stuttering.

Proof: It can easily be shown by induction on the structure of the formula that every formula that only uses the modality $\mathcal{U}$ expresses a property that is closed under stuttering. The proof of the lemma follows from the above observation and theorem 3.1. ■

Note that the formula expressing the simple safety property $safe$, which is a safety property with stuttering, only uses the temporal operator $\mathcal{U}$.

Consider a complete axiomatization for the set of valid formulae in PTL. It can easily be shown that all the formulae belonging to the set T generated by the following rules express safety properties with stuttering.

- Every positive formula built using only the modality $\mathcal{U}$ is in T;
- If $\vdash f \leftrightarrow g$ and $g \in T$ then $f \in T$.

3.3 Strong Safety Properties

In this section, we give an exact characterization of strong safety properties expressible in PTL. We prove a stronger result showing that the class of strong safety properties definable by finite state automaton are exactly the class of properties expressed by positive formulae that only use the temporal operator $\Box$. First, we need the following definitions.

A finite state automaton A on infinite strings is a 5-tuple $(\Delta, Q, \delta, I, F)$ where Δ is the alphabet, Q is the set of automaton states, $\delta : (Q \times \Delta) \rightarrow 2^Q$, $I \subseteq Q$ is the set of start states, $F \subseteq Q$ is the set of final states. Let $t \in \Delta^\omega$. A run of A on t is a ω-sequence of states $q_0, q_1, ...$ such that $q_0 \in I$ and $\forall i \geq 0, q_{i+1} \in \delta(q_i, t_i)$. The sequence t is said to be accepted by A iff there exists a run of A on t that contains a state from F repeated infinitely often. Consider an automaton A whose alphabet is the set Σ. From now onwards, we only consider such automata. We say that a property C is defined by A if C is exactly the set of infinite strings accepted by A. We let $L(A)$ denote the property defined by A.

For a finite sequence s of elements, we let $length(s)$ denote the number of elements in s, and for an infinite sequence s we define $length(s)$ to be ∞. Given two sequences $s = (s_0, s_1, ...)$ and $t = (t_0, t_1, ...)$, we say that t is a subsequence of s if there exists integers $i_0 < i_1 < ...$ such that for all p such that $0 \leq p < length(t)$, $t_p = s_{i_p}$.

The following theorem characterizes the class of strong safety properties definable by finite state automata.

THEOREM 3.3: The class of strong safety properties defined by finite state automata is exactly the class of properties expressed by positive formulae using only the temporal operator $\Box$.

The above theorem follows from the lemmas 3.4 and 3.6 established below. Lemma 3.5 shows that every positive formula that only uses the operator $\Box$ expresses a strong safety property. Lemma 3.6 shows that, for every strong safety property definable by an automaton, there is a positive formula using the $\Box$ operator that expresses the property.

It is well known (see [Si83]) that for every property expressed by a PTL formula there exists an automaton on infinite strings that defines the property. From this and lemma 3.6, it follows that every strong safety property expressed by a PTL formula is also expressed by a positive formula using the $\Box$ operator. This give us the following corollary.

COROLLARY 3.4: The class of strong safety properties expressed by PTL formulae is exactly the class of properties expressed by positive formulae using only the temporal operator $\Box$.

LEMMA 3.5: Every positive formula formed using only $\Box$ expresses a strong safety property.

Proof: Let f be a positive formula formed using $\Box$. It follows from theorem 3.2 that f expresses a safety property with stuttering. We have to prove that for any t, if $t, 0 \models f$ and t' is a sequence obtained by deleting any non-initial state then $t', 0 \models f$. We prove this by induction on the structure of f. This is trivially true if f does not have $\Box$. The induction step is easily seen for the case when f is of the form $g \wedge h$ or is of the form $g \vee h$. Now let $f = \Box g$. In this case, $\forall j \geq 0, t, j \models g$. Using this fact and the induction hypothesis for g, it is easily seen that $t, 0 \models f$. ■

Note that the formula that expresses the property *monotone-p* , given in section 2.2, only uses the $\Box$ operator.

Now, we want to show that every strong safety property definable by a finite state automaton on infinite strings can be expressed by a positive formula using the $\Box$ operator. From this result, it would follow that the class of strong safety properties that can be expressed in PTL is exactly those properties that are expressed by positive formulae using the $\Box$ operator.

LEMMA 3.6: If C is an automaton definable strong safety property then there exists a positive formula built using $\Box$ that expresses C.

Proof: Let $A = (\Sigma, Q, \delta, I, F)$ be an automaton that accepts C. Corresponding to A, we define a directed graph $G = (V, E)$ as follows: $V = (Q \times \Sigma) \cup \{v_0\}$ and $E = \{((q, a), (q', a')) : q' \in \delta(q, a')\} \cup \{(v_0, (q', a')) : q' \in \delta(q, a') \text{ for some } q \in I\}$. Notice that the vertex v_0 is a source vertex, i.e. no edges enter the vertex. Each vertex other than v_0 corresponds to a state of the automaton and an input from Σ. Each edge in E corresponds to a transition of the automaton on an input symbol. Let $p = p_0, p_1, ...$ be an infinite sequence of nodes of G where $p_0 = v_0$ and $\forall i > 0, p_i = (q_i, a_i)$. Define a sequence $t(p) = (t_0, t_1, ...) \in \Sigma^\omega$ where $\forall i \geq 0, t_i = a_{i+1}$. The sequence p is a path if $\forall i \geq 0, (p_i, p_{i+1}) \in E$. p is said to be *accepting* iff there exist infinitely many values of i such that $q_i \in F$. Let $L(G) = \{t(p) \in \Sigma^\omega : p \text{ is an accepting path in } G \text{ starting from } v_0\}$. It is easily seen that $L(A) = L(G)$. If C is empty then any unsatisfiable propositional formula expresses C. So, assume that C is non-empty. Let V' be the set of nodes in G such that for each node in V' there exists an accepting path starting from v_0 that passes through this node. Note that $v_0 \in V'$. Let $G' = (V', E')$ where $E' = \{(r, s) : r, s \in V' \text{ and } (r, s) \in E\}$. It is easily seen that $L(G') = L(G) = L(A) = C$. Since C is a safety property the following claim is easily seen.

Claim 1: $C = \{t(p) : p \text{ is a path in } G' \text{ starting from } v_0\}$. ■

Claim 2: If $p = p_0, p_1, ...$ is any infinite sequence of nodes in G' such that $p_0 = v_0$, $(v_0, p_1) \in E'$ and every prefix of p is a subsequence of an infinite path in G' starting from v_0 then $t(p) \in C$.

Proof: Let p be as defined in the statement of the claim. Clearly, every prefix of $t(p)$ is a subsequence of $t(q)$ for some infinite path q in G' starting from v_0. Since C is a strong safety property it is easily seen that every prefix of $t(p)$ is a prefix of some sequence in C. Since C is a safety property, it follows that $t(p) \in C$. ■

Let $G'' = (V'', E'')$ be a directed graph where V'' is the set of strongly connected components[2] of G' and $(C_1, C_2) \in E''$ iff there exists an edge in G' from some node in C_1 to some node in C_2. Since v_0 is a source vertex, it is the case that the singleton $\{v_0\}$ is a strongly connected component and is in V''. Clearly, G'' is acyclic. Hence there are only a finite number of paths in G''. Let $c = (q, a) \in V' - \{v_0\}$. With c we associate a propositional formula $g(c) = g_1 \wedge g_2$ where g_1 is the conjunction of all atomic propositions P such that $a(P) = \mathbf{true}$ and g_2 be the conjunction of all $\neg P$ such that $a(P) = \mathbf{false}$ (recall that $a \in \Sigma$). For any $C \in V''$, let $g(C)$ be the disjunction of all $g(c)$ such that $c \in C$. Let $d = C_0, C_1, ..., C_m$ be a path in G''. We define temporal formulae $h_m, h_{m-1}, ..., h_0$ inductively as follows:

- $h_m = \Box g(C_m)$,
- $h_i = \Box(g(C_i) \vee h_{i+1})$ for $0 \le i < m$.

Assume C_0, the first node in d, has the property that $C_0 \neq \{v_0\}$ and there is an edge in G' from v_0 to at least one node in C_0. We call such a path a *required* path. Let D be the set of all nodes x in C_0 such that there is an edge from v_0 to x. Let $g(d) = g(D) \wedge h_0$. Let f be the disjunction of all $g(d)$ such that d is a required path in G''.

Claim 3: For any $t \in \Sigma^\omega$, $t, 0 \models f$ iff $t \in C$.

Proof: Assume that $t \in C$. From claim 1, it follows that there exists a path p in G' starting from v_0 such that $t = t(p)$. From this and the way we constructed the formula f, it follows that $t, 0 \models f$. Now, let $t, 0 \models f$. It is easily seen that there exists a $p = p_0, p_1, ...$ which is an infinite sequence of nodes in G' such that $p_0 = v_0, (v_0, p_1) \in E'$ and every prefix of p is a subsequence of an infinite path in G' and that $t = t(p)$. From claim 2, it follows that $t \in C$. ■

It is clear that the formula f defined above is a positive formula using only the modality $\Box$. ■

It is to be noted the proof of Lemma 3.6 is constructive. The formula f generated above has a special form. It is a disjunction of clauses of the form $p \wedge h$ where p is a propositional formula and h is of the form h_0 given in the proof of lemma 3.6. This establishes a normal form for strong safety formulae.

4 Liveness Properties in Temporal Logic

4.1 Absolute Liveness

LEMMA 4.1: A formula f in PTL expresses an absolute liveness property iff f is satisfiable, and f and $\Diamond f$ are expressively equivalent.

Proof: Assume that f expresses an absolute liveness property. By definition, f is a satisfiable formula. We will show that f and $\Diamond f$ are expressively equivalent. Clearly, any

[2] Recall that a strongly connected component is maximal set of vertices such that every two vertices in it lie on a cycle, or is singleton set containing a node that has a self loop and does not lie on any other cycle, or is a singleton set containing a node which does not lie on any cycle.

Property	Type of Formulas
Safety	Positive formulas using $\mathcal{U}$ and $\bigcirc$
Safety with Stuttering	Positive formulas using only $\mathcal{U}$
Strong Safety	Positive formulas using only $\Box$
Invariant	$\Box f$ where f is propositional
Absolute Liveness	Satisfiable f such that $\vdash f \leftrightarrow \Diamond f$
Stable	Satisfiable f such that $\vdash f \leftrightarrow \Box f$
Fairness	Satisfiable f such that $\vdash f \leftrightarrow \Diamond f$ and $\vdash f \leftrightarrow \Box f$

Table 1: Table of Properties and type of formulas that express them

infinite sequence of states that satisfies f also satisfies $\Diamond f$. Now, consider any $t \in \Sigma^\omega$ that satisfies $\Diamond f$. For some $i \geq 0$, $t, i \models f$. Since, the property expressed by f is an absolute liveness property, it is the case that $t = t(i-1)t(i, \infty)$ also belongs to this property. Hence, t also satisfies f.

Now, assume that f is satisfiable, and f and $\Diamond f$ are expressively equivalent. Consider any $t \in \Sigma^\omega$ that satisfies f and u be any finite sequence in Σ^*. Clearly, the sequence ut satisfies $\Diamond f$. Since, f and $\Diamond f$ are expressively equivalent, it follows that ut also satisfies f. From this, we see that the property expressed by f is an absolute liveness property. ■

The formula $\Diamond$*terminate* expresses the program termination property. Note that the formula expressing the liveness property for resources *live-resource* , defined in subsection 2.2, satisfies lemma 4.1.

4.2 Fairness Properties

The following lemma characterizes stable properties. It trivially follows from the definition of stable properties.

LEMMA 4.2: A formula f in PTL expresses a stable property iff f is satisfiable , and f and $\Box f$ are expressively equivalent. ■

Some examples of formulae that express stable properties are $\Box P$, $\Box(P \vee \Box Q)$ and $\Box\Diamond P$.

We have already shown that f expresses an absolute liveness property iff f is satisfiable and the formulas f and $\Diamond f$ are expressively equivalent. The following lemma, i.e. LEMMA 4.3, follows from the above results and the definition of fairness properties. Recall that a fairness property is defined to be a property that is both an absolute liveness property as well as a stable property.

LEMMA 4.3: A formula f expresses a fairness properties iff f is satisfiable and $f, \Diamond f$ are expressively equivalent and $f, \Box f$ are expressively equivalent. ■

Note that the formula, given in section 2.2, which expresses the channel fairness property, satisfies the above lemma.

The property *channel-fairness* is usually called a strong fairness property. A weaker notion of channel fairness, usually called *weak channel fairness*, is expressed by the following formula:

$\Diamond\Box send \rightarrow \Box\Diamond receive$

Another important fairness property is *process fairness*. If the atomic proposition ex asserts that the present state is reached by the execution of a step of process 1, then $\Box\Diamond ex$ asserts that process 1 is executed infinitely often. This is process fairness for process 1.

5 Complexity and Other Issues

So far, we have considered syntactic characterization of formulas that express safety and liveness properties. In this section, we present simple decision procedures for recognizing safety formulae and liveness formulae. Table 2, given at the end of the section, summarizes the results of this section.

We first consider non-deterministic automata that define safety properties and give some simple properties of such automata.

THEOREM 5.1: Let A be any non-deterministic automaton then $L(A)$ is a safety property iff the following condition is satisfied: There exists a partition {*good, bad*} of the set of states of A such that any t is accepted by A iff there exists a run of A starting from an initial state such that the run contains only *good* states.

Proof: Let *good* be the set of states such that there exists an accepting run starting from this state on some input. Let the set *bad* consist of the remaining states. It should be straightforward to verify the theorem for these sets of states. ■

Next, we present an algorithm that decides if a given PTL formula is a safety formula. For a given PTL formula f, we define $SF(f)$ to be the set consisting of the subformulae of f or the negations of subformulae of f. A subset $c \subseteq SF(f)$ is said to be *complete* if the following two conditions are satisfied: (i) for each $g \in SF(f)$, exactly one of g and $\neg g$ is in c; (ii) for $g = g_1 \wedge g_2 \in SF(f)$, $g \in c$ iff $g_1 \in c$ and $g_2 \in c$. The algorithm that we present constructs a directed graph, $tab(f) = (V, E)$, where V is the set of all complete subsets of $SF(f)$. The set of edges E is defined as follows. The pair $(c, d) \in E$ iff the following conditions are satisfied:

- for any $g = g_1 \ \mathcal{U} \ g_2 \in SF(f)$, $g \in c$ iff either $g_2 \in c$, or $g_1 \in c$ and $g \in d$;
- for any $g = \bigcirc g_1 \in SF(f)$, $g \in c$ iff $g_1 \in d$.

The construction of $tab(f)$ has been given in many of the earlier works (see [Si83, LPZ85] for details). For any $c \in V$, we say that c is a *good* node if the conjunction of all subformulae in c is satisfiable. Also, for any $c \in V$, define $\pi(c)$ to be the state[3] such that for any atomic proposition $P \in SF(f)$, $\pi(c)(P) =$**true** iff $P \in c$. For a finite or infinite path $p = p_0, p_1, ...$ in $tab(f)$, we let $\pi(p)$ to be the sequence of states $\pi(p_0), \pi(p_1), ...$ Consider a finite path $p = (p_0, p_1, ..., p_k)$ in G such that the node p_0 contains a subformula g of the form $\neg(g_1 \mathcal{U} g_2)$. We say that g is fulfilled on p if for some i, such that $0 \leq i \leq k$, $\neg g_1 \in p_i$. An infinite path in $tab(f)$ is said to be *accepting* if for every node c on the path and for every subformula of the form $g = \neg(g_1 \ \mathcal{U} \ g_2) \in c$, the subformula either $\neg g_1 \in c$ or there exists a future node d after c on the path such that $\neg g_1 \in d$. The following lemma has been proved in many of the earlier works (see [Si83, LPZ85]).

LEMMA 5.2: $tab(f)$ satisfies the following properties:
(a) For any node c, the conjunction of all subformulae in c is satisfiable iff there exists an accepting infinite path starting from c;
(b) For any $t \in \Sigma^\omega$, $t, 0 \models f$ iff there exists a node c such that $f \in c$ and there is an infinite accepting path p starting from c such that $t = \pi(p)$. ■

LEMMA 5.3: For every $t \in \Sigma^\omega$ satisfying the property $\forall i \geq 0 \exists u \in \Sigma^\omega$ such that $t(i)u, 0 \models f$, there exists an infinite path p in $tab(f)$ starting from a node that contains f and that contains only good nodes and such that $\pi(p) = t$.

[3]Recall that a state maps the set of atomic propositions to the set { **true, false**}

Proof: Let $t \in \Sigma^\omega$ satisfy the property that $\forall i \geq 0 \exists u \in \Sigma^\omega$ such that $t(i)u, 0 \models f$. From lemma 5.2, it follows that for each $i \geq 0$, there exists an accepting path α_i in tab(f) starting from a node that contains f and such that $\pi(\alpha_i(i)) = t(i)$. Clearly, for each i, α_i only contains good nodes. By simple induction, it can be shown that there exists an infinite sequence of finite paths β_0, β_1,...,β_j,... of increasing lengths and each starting from a node that contains the formula f and satisfying the following properties: for each $j \geq 0$, β_j is a prefix of α_i for infinite number of values of i; each β_j is a prefix of β_{j+1}. Now, consider the infinite path which is the limit of the above sequence of finite paths. This path satisfies the required property. ■

From the definition of *tab*(f), it should be clear that *tab*(f) is identical to *tab*($\neg f$). Let *tab*$(f) = (V, E)$. Now, we define another directed graph, called *cross-tab*(f) which is a cross product of *tab*(f) with itself. This graph will be used in the decision procedure for recognizing safety formulae. Since *tab*(f) and *tab*($\neg f$) are identical, it is the case that *cross-tab*(f) is the cross product of *tab*(f) and *tab*($\neg f$). Formally, *cross-tab*$(f) = (V', E')$ where $V' = \{(c, c') \in V \times V : \pi(c) = \pi(c')\}$, $E' = \{((p, q), (p', q')) : (p, p'), (q, q') \in E\}$. Let $(p_0, q_0), (p_1, q_1), ..., (p_k, q_k)$ be a finite path in *cross-tab*(f). Then, it should be easy to see that $p_0, p_1, ..p_k$ and $q_0, ..., q_k$ are finite paths in *tab*(f). We say that the above path in *cross-tab*(f) is fulfilling iff the path $p_0, p_1, ..., p_k$ is a fulfilling path in *tab*(f), i.e. for every formula of the form $\neg(g_1 \; \mathcal{U} \; g_2)$ in p_0 there exists a node (p_i, q_i), $0 \leq i \leq k$, such that $\neg g_1$ is in p_i. Note that the fulfillment of a path in *cross-tab*(f) is defined with respect to its projection on the first component of each node.

LEMMA 5.4: A formula f is not a safety formula iff there exist nodes (p, q) and (p', q') in V' such that the following conditions are satisfied:

- (a) $\neg f \in p, f \in q$, and (p', q') is reachable from (p, q) in *cross-tab*(f);
- (b) (p', q') lies on a fulfilling cycle;
- (c) q and q' are good nodes.

Proof: The proof of the reverse implication is easily seen from the following argument. Assume (a),(b),(c). There exists a path of the form $\alpha\beta^\omega$ in *cross-tab*(f) starting from (p, q) where β is the cycle on which (p', q') lies. Let γ and γ' be the projections of the above path on the first and second components respectively. Clearly, γ is an accepting path in *tab*($\neg f$) and starts from a node that contains $\neg f$, and hence from lemma 5.2 (b), it is the case that $\pi(\gamma), 0 \models \neg f$. Since q' is a good node, it is the case that all the nodes on γ' are good nodes of *tab*(f). This is due to the fact in *tab*(f) any node that has a path to a good node, is also a good node. Since γ' starts from a node that contains f and all the nodes on γ' are good nodes, it follows from lemma 5.2 that each prefix of $\pi(\gamma)$ can be extended to an infinite sequence that satisfies f. Since $\pi(\gamma) = \pi(\gamma')$, it follows that f is not a safety formula.

Assume that f is not a safety formula. There exists $t \in \Sigma^\omega$ such that t satisfies $\neg f$ and every prefix of t can be extended to an infinite sequence to satisfy f. For each $i \geq 0$, let r_i be the set of subformulae of f or their negations that are satisfied by $t(i, \infty)$. Let $r = r_0, r_1, ...$ It can easily be seen that r is an accepting infinite path in *tab*($\neg f$) and $\neg f \in r_0$. Using lemma 5.3, we see that there exists an infinite path $s = s_0, s_1, ...$ in *tab*(f) such that $f \in s_0$, $\pi(s) = t$ and s contains only good nodes. Note that while r is an accepting path, s need not be one. Now, it should be easy to see that the sequence $(r_0, s_0), (r_1, s_1), ..., (r_i, s_i), ...$ is an infinite path through *cross-tab*(f). From this and the properties of the paths r and s, it should be easy to see that there exist integers i, j such that $j \geq i$, $r_i = r_j$, $s_i = s_j$, and for every subformula of the form $\neg(g_1\ \mathcal{U}\ g_2)$ in r_i $\exists k$ such that $i \leq k \leq j$ and $\neg g_1 \in r_j$. Now take $p = r_0$, $q = s_0$, $p' = r_i$ and $q' = s_i$. It should be easy to see that p, q, p' and q' satisfy (a),(b) and (c). ■

.

THEOREM 5.5: The set of safety formulae in PTL is PSPACE-complete.

Proof: Let f be a PTL formula. We give an algorithm which checks if f is a safety formula or not. The algorithm builds *cross-tab*(f) and checks for conditions (a),(b), (c) of lemma 5.4. Such an algorithm will take time exponential in the length of f. This algorithm also uses exponential space. Instead of building *cross-tab*(f), the algorithm can guess a path through *cross-tab*(f), and using an approach similar to the satisfiability algorithm for PTL given in [SC85] it can check if f is a safety property or not. The resulting algorithm uses space which is only polynomial in the length of f.

The set of safety formulae is PSPACE-hard due to the following reduction from the set of valid formulae in PTL. Any formula f is valid iff $f \vee \Diamond Q$ is a safety formula where Q is an atomic proposition that does not appear in f. ■

We describe below an algorithm which given a formula f decides if f expresses a liveness property. First build *tab*(f). From *tab*(f) delete all those nodes from which there is no infinite accepting path. Let the resulting structure be the *reduced-tab*(f). Consider an non-deterministic automaton $A(f)$ on finite strings defined as follows. The states of $A(f)$ include all the nodes of *reduced-tab*(f) together with an additional special state *init*. The input alphabet is Σ. There is a transition from state s to s' on input a if there is an edge in *reduced-tab*(f) from s to s' and for any atomic proposition $P, a(P) = \mathit{true}$ iff $P \in s'$. There is a transition from from *init* to a state s on input a iff $f \in s$ and $a(P) = \mathit{true}$ iff $P \in s$. All states excepting the start state *init* are final states of $A(f)$. It is easily seen that f expresses a liveness property iff $A(f)$ accepts all the strings in Σ^*. The later problem can easily be decided in time exponential in the size of $A(f)$ and space polynomial in the size of $A(f)$. Thus, the above algorithm takes time double exponential in the length of f and space exponential in the length of f.

THEOREM 5.5: The above algorithm decides if a given formula expresses a liveness property. ■

Property	**Complexity of Recognition Alg**
Safety	PSPACE-complete
Liveness	Double Exponential

Table 2: Table of Properties and complexity of the Recognition Algorithms

6 Conclusions

In this paper, we have considered the problem of characterizing safety and liveness properties in temporal logic. We have shown that all the positive formulae formed using $\mathcal{U}$ and $\bigcirc$ express safety properties and those formed using just only $\mathcal{U}$ express safety properties with stuttering. We have shown that the properties expressed by positive formulae using only the operator $\Box$ is exactly the set of strong safety properties that can be expressed in PTL. We have also given such characterizations for absolute liveness and fairness properties. In addition, we have also presented decision procedures for recognizing safety and liveness formulae.

Although we have not given completeness results for safety properties, it can be shown that the set of safety properties expressible in PTL are exactly those expressed by positive formulae using $\mathcal{U}$ and $\bigcirc$. The proof of this result is rather complex, and for this reason we have not stated this result as part of this paper. The major steps of this proof can be stated as follows. Let f be an arbitrary safety formula in PTL and P be the property that f expresses. We like to show that there exists a positive formula that expresses the same property. Let $Pr(P)$ be the set of prefixes of the sequences in P. $Pr(P)$ is a set of finite sequences, and from known results (see [SCFM] for references), it can easily be shown that $Pr(P)$ is a star-free regular set. Again, from known results it follows that there exists a cascade product A of reset automata (see [SCFM] for definitions) that recognizes $Pr(P)$. The automaton A is deterministic and the set of finite sequences it recognizes is prefix closed. Therefore, we can obtain a partition $\{good, bad\}$ of the states of A such that the initial state is a good state, and A accepts a finite sequence iff it always remains in *good* states when run on that input. The acceptance condition of A can be trivially extended to infinite sequences. We say that A accepts an infinite string iff it always remains in *good* states when run on this input. It is easy to see that the set of infinite sequences accepted by A is the set P. The second step of the proof shows that we can obtain a positive PTL formula that expresses the set of infinite sequences accepted by A. This part of the construction is very complex. Using the same approach as above, we can also show that the set of safety properties with stuttering that can be expresses in PTL are exactly those expressed by positive formulae using only the $\mathcal{U}$ operator. It will be interesting to investigate if we can give simpler proofs of these results.

The logic that we have used in this paper is the future logic. It only uses the future temporal operators. It is shown in [LPZ85] that there is a simple characterization of safety properties if we use past operators. There, it was shown that all safety properties expressible in PTL are expressible by formulae of the form $\Box\varphi$ where φ is a formula that only uses past operators. In [MP90], a hierarchy of other interesting temporal properties have been presented. The problems of recognizing safety and liveness properties given by automata were considered in [AS86] and algorithms for these problems were presented there.

In [MP89], a complete proof method for proving safety properties of programs expressed in past temporal logic has been given. It will be interesting to investigate such proof methods for safety properties expressed in future temporal logic without converting the formula into a formula that uses past operators. Also, giving a characterization of liveness properties in full PTL is still an open problem.

References

[ADS86] B. Alpern, A. J. Deemers, F. B. Schneider, Safety without Stuttering, Information Processing Letters 23(4):177-180.

[AS85] B. Alpern, F.Schneider, Defining Liveness, Information Processing Letters, 21:181-185.

[AS86] B. Alpern, F. B. Schneider, Recognizing Safety and Liveness, TR 86-727, Computer Science Department, Cornell University, Jan 1986.

[CES86] E. M. Clarke, E. A. Emerson, A. P. Sistla, Automatic Verification of finite-state Concurrent Systems using Temporal Logic Specifications, ACM Transactions on Programming Languages and Systems 8(2):244-263.

[Em83] Emerson E. A., Alternative Semantics for Temporal Logic, Theoretical Computer Science, Vol 26, pp 121-130, 1983.

[EL87] E. A. Emerson, C. L. Lei, Modalities for Modelchecking: Branching Time Strikes Back, Science of Computer Programming, Vol 8, pp 275-306, 1987.

[Fr86] N.Francez, Fairness, Texts and Monographs in Computer Science, Springer-Verlag 1986.

[La77] Proving Correctness of Multiprocess Programs, IEEE Transactions on Software Engineering, SE-3, 2:125-143.

[La85] L. Lamport, Logical Foundation, Distributed Systems- Methods and Tools for Specification, Vol 190, Lecture Notes in Computer Science, Springer-Verlag, Berlin,

[LPZ85] O.Lichtenstein,A.Pnueli,L.Zuck, The Glory of the Past, Lecture Notes in Computer Science, 193, Proceedings of the workshop on Logics of Programs, Brookline College, June 1985.

[MP89] Z. Manna and A. Pnueli, Completing the Temporal Picture, Proceedings of the 16th International Colloquium on Automata, Languages and Programming, 1989, Also appeared in Theoretical Computer Science, 1991, 83(1):97-130.

[MP90] Z. Manna and A. Pnueli, A hierarchy of Temporal Properties, Proceedings of the 9th ACM Symposium on Principles of Distributed Computing, 1990, pp 377-408.

[MP92] Z. Manna and A. Pnueli, The Temporal Logic of Reactive and Concurrent Systems— Specification, Springer-Verlag, 1992.

[OL82] S.Owicki, L.Lamport, Proving Liveness Properties of Concurrent Programs, ACM Transactions on Programming Languages and Systems 4,No.3, 1982.

[Pn77] A.Pnueli, The Temporal Logic of Programs, Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, Providence, RI(1977).

[SCFM] A.P. Sistla, E. M. Clarke, N. Francez, A. R. Meyer, Can Message Buffers be Axiomatized in Temporal Logic?, Information and Computation, 63(1,2):88-112.

[SC85] A.P.Sistla,E.M.Clarke, Complexity of Propositional Temporal Logics, Journal of the Association for Computing Machinery, Vol.32,No.3, July 1985.

[Si83] A.P.Sistla, Theoretical Issues in the Design and Verification of Distributed Systems, Ph.D. thesis 1983, Harvard University.

[Si85] A.P.Sistla, On Characterization of Safety and Liveness Properties in Temporal Logic, Proceedings of the 4th ACM Symposium on Principles of Distributed Computing, August, 1985, Minaki, Canada.

[Si86] A. P. Sistla, Characterization of Safety and Liveness Properties in Temporal Logic, GTE Laboratories Technical Report, 1986.

[Tho86] W.Thomas, Safety and Liveness Properties in Propositional Temporal Logic: Characterization and Decidability, Schriften Zur Informatik, Bericht Nr. 116. April 1986.