October 1987

Report No. STAN-CS-87-1186

A Hierarchy of Temporal Properties

by

Zohar Manna and Amir Pnueli

Department of Computer Science

Stanford University
Stanford, CA 94305

October 1987

A Hierarchy of Temporal Properties

Zohar Manna.
Stanford University
and
Weizmann Institute of Science

Amir Pnueli
Weizmann Institute of Science

Abstract

We propose a classification of temporal properties into a hierarchy which refines the known *safety-Ziveness* classification of properties. The new classification recognizes the classes of *safety, guarantee, persistence, fairness,* and *hyper-fairness.* The classification suggested here is based on the different ways a property of finite computations can be extended into a property of infinite computations. For properties that are expressible by temporal logic and predicate automata, we provide a syntactic characterization of the formulae and automata that specify properties in the different, classes. We consider the verification of properties over a given program, and provide a. unique proof principle for each class.

0. Introduction

Reactive systems are systems whose function is to maintain some continuous interaction with their environment. Such systems should be specified and analyzed in terms of their behaviors, i.e., the sequences of st ates or events they generate during their operation. We may view a reactive program as a generator of *computations*, which are finite or infinite sequences of states or events.

The work of the first author was supported by the National Science Foundation under Grant DCR-84-13230 and by the Defense Advanced Research Projects Agency under Contract N00039-84-C-0211. The work of the second author was done during a visit, to the University at Austin which is gratefully acknowledged.

In general, we define a *property* as a set of computations. A program *P* is said to have the property II if all the computations of *P* belong to II. Several languages and formalisms have been proposed for expressing properties of programs, including the language of temporal logic and the formalism of predicate automata.

A useful partition of properties into the classes of *safety* and *Ziveness* properties has been suggested by Lamport in [L]. An important advantage of this classification is that each class encompasses properties of similar character. For example, safety properties characteristically represent requirements that should be continuously maintained by the system. They often express *invariance* properties of a system. Liveness properties, on the other hand, characteristically represent requirements that need not hold continuously, but whose eventual realization must be guaranteed. They often express the *progress* properties of a system. A complete specification of a system must include properties of both classes.

To draw an analogue from sequential terminating programs, safety properties correspond to partial correctness, which does not guarantee termination but only that all terminating computations produce correct results. Liveness properties correspond, according to our view, to total correctness which also guarantees termination. For reactive systems, which may never terminate, the role of liveness properties is even more important than that of termination for sequential programs.

While it is generally recognized that a complete specification of a system should include both a safety and a liveness part, there is an additional cost, in a language that, can express both classes of properties. For example, if we are ready to restrict ourselves to expressing only safety properties, then the relatively simpler language of predicates over *finite* behaviors suffices. The only justification for using temporal logic, which is a considerably more expressive and consequently more complex language, is for expressing liveness properties. Similarly, if we use automata or transition systems for specification, and restrict ourselves to safety properties, it is sufficient to consider automata over *finite* inputs. Only when we want to express liveness properties do we have to use automata over infinite inputs. Thus, a major justification for studying the classification of properties is to identify the tradeoff between completeness of specification and complexity of the specifying language.

Another reason for the distinction between the classes is that their verification calls for different proof principles. To establish a safety or invariance property, we show that it is initially true and that it is preserved by each individual action of the program. To establish a liveness property we usually employ induction on the distance from the realization of the goal guaranteed by the property.

A formal characterization of the two classes has been given by Alpern and

Schneider [AS1]. Let Σ denote the set of states that may appear in computations, Σ^+ denote the set of all non-empty finite computations, Σ^ω denote the set of all infinite computations, and $\Sigma^\infty = \Sigma^+ \cup \Sigma^\omega$ denote the set of all finite and infinite non-empty computations.

A property $\Pi \subseteq \Sigma^\omega$ is defined in [AS1] to be a safety property if for every computation $\sigma \in \Sigma^\omega$:

$$\sigma \in \Pi \leftrightarrow \forall\sigma'(\sigma' \prec \sigma): \exists\sigma''(\sigma'' \in \Sigma^\omega): \sigma' \cdot \sigma'' \in \Pi,$$

where $\sigma' \prec \sigma$ denotes that σ' is a finite prefix of σ and $\sigma' \cdot \sigma''$ denotes the concatenation of σ' and σ''. This definition means that a computation belongs to Π iff all its finite prefices can be extended to computations in Π.

A property $\Pi \subseteq \Sigma^\omega$ is defined to be a liveness property if:

$$\forall\sigma(\sigma \in \Sigma^+): \exists\sigma'(\sigma' \in \Sigma^\omega): \sigma \cdot \sigma' \in \Pi.$$

That is, every finite computation can be extended to a computation in Π.

Sistla [S] gave a syntactic characterization of the temporal formulae that specify safety properties. They are all the formulae that can be built up of propositions, the positive boolean operators ($\vee$ and $\wedge$). and the unless operator (weak *until).* He also gave a characterization of formulae expressing some restricted classes of liveness properties.

Some consequences of the definition of safety and liveness, as given in [AS1] and syntactically characterized in [S], are that the two classes are essentially disjoint'. Only trivial properties such as F (the empty set of computations) and T (Σ^∞) belong to the intersection of the two classes. For example, in general, a disjunction or a conjunction of a. safety property and a liveness property is neither a safety nor a liveness property. If we intend to base our verification approach on proof principles appropriate to each of the classes, then there are some properties, such as the combinations of safety and liveness mentioned above, for which there ase no directly applicable rules.

Mainly motivated by these verification considerations, a. different definition of the safety-liveness classification was presented in [LPZ]. The classification proposed in that paper is a hierarchy rather than a. partition, and is based on the syntactic form of the temporal formulae expressing properties in these classes. The classification is the following:

A *safety* property is a property specified by a temporal formula, of the form $\Box\, p$, for some *past* formula p. ([LPZ] uses temporal logic with past operators, and a past formula,' is any formula containing no future operators.) A *basic liveness* property is a property specifiable by a formula of one of the forms $\Diamond p$, $\Box$ Op, and $\Diamond\Box p$, for some past formula p. A liveness property is a positive boolean combination of basic liveness properties.

Also established in [LPZ] is the fact that every temporal formula is equivalent to a positive boolean combination of the basic liveness formulae, making the. liveness class include all the properties specifiable in temporal logic.

This classification views liveness as an *extension* of safety properties, and explains why the proof rule for liveness properties has to be an extension of the proof rule for safety. It also ensures that every specifiable property has a proof rule adequate for its verification.

To further clarify the differences between the classification-as-partition approach, represented in [AS1]–[AS3] and [S], and the classification-as-hierarchy approach, represented in [LPZ] and this paper, let us consider again the properties of partial correctness, termination, and total correctness. Obviously, total correctness is the conjunction of partial correctness and termination. Both approaches agree on classifying partial correctness as a safety property. The partition approach classifies termination as liveness, but classifies total correctness as a conjunction of a safety property and a liveness property. The hierarchy approach classifies both termination and total correctness as liveness properties.

In the present paper we study in greater detail the different classes of properties. We refine the hierarchy by identifying as sepasate subclasses the properties expressible by the three basic forms of liveness formulae. We study the inclusion relations between these classes and their closure properties under union and intersection. As justification for the distinction between these classes, we mention typical examples of properties falling into each of the classes.

We study the proposed classification from three distinct viewpoints. First, we consider a semantic definition of the classes, not considering any particular formalism for their specification. Next, we consider properties that are expressible in temporal logic, and give a syntactic characterization of the formulae expressing properties in each class. Last, we consider the specification of properties by predicate automata. Again we give a characterization of the classes by restrictions on the automata expressing them.

A hierarchy, very similar to the one considered here, has been studied extensively in the context of automata over infinite words, which is the third view we consider. The properties of the lower ranks of the hierarchy, which are our main subject of interest, have been established by Landweber in [Lan]. The complete hierarchy has been analyzed by Wagner in [Wag], and several years later in [Kam]. Consequently, many of the technical results described in the section on automata have been established in these two works. The similar results about temporal logic can usually be derived from the automata results by restriction to non-counting automata ([Z1]). Indeed, the characterization of the temporal logic hierarchy, and the fact that it is a strict hierarchy, have been recently obtained by Zuck ([Z2]).

Outline of the Paper

In Section 1, we present the semantic view of the classification. We introduce two operators that generate infinitary properties out of finitary properties, and base the classification on the combination of operators necessary to construct properties in each of the classes. Some closur properties of the classes are studied.

In Section 2, we restrict our attention to infinitary properties that are specifiable by temporal logic. For each class, we specify a syntactical restriction on the formulae that define properties in this class. We show that, up to equivalence, the syntactically restricted formulae possess the appropriate closure properties of each class.

In Section 3, we present the predicate automata as a formalism for specifying infinitary properties. For each class, we specify a structural restriction on the automata that define properties in this class. We prove that a property, which can be specified by an automaton, belongs to one of the classes (according to the semantic definition) iff it can be specified by an automaton that obeys the structural restrictions associated with the class. A similar result is established for the syntactical restrictions imposed on temporal logic formulae.

In Section 4, we establish the connection between specifications by temporal logic and specifications by automata.

In Section 5, we list proof principles for the various classes.

1. Semantic View

The main issue in the safety-liveness dichotomy, according to the semantic view, is how we can extend properties of *finite* computations into properties of *infinite* computations.

For a finite computation $\sigma \in \Sigma^+$ and a computation $\sigma' \in \Sigma^\infty$, we denote by $\sigma \prec \sigma'$ the fact that σ is a finite prefix of σ' but different from σ' (a proper finite prefix). We denote by $\sigma \preceq \sigma'$ the more general relation $(a \prec a') \vee (a = \sigma')$.

Properties II $\subseteq \Sigma^+$ are referred to as *finitary* properties, while properties II $\subseteq \Sigma^\omega$ are referred to as *infinitary* properties. For a property II $\subseteq \Sigma^\infty$, we denote by $Pref(\Pi)$ the set of all *finite* prefices of II:

$$Pref(\Pi) = \{\, \sigma \in \Sigma^+ \mid \sigma \preceq \sigma' \text{ for some } \sigma' \in \text{II}\}.$$

We denote by $\sigma[0\,..\,k]$ the finite prefix $s_0, \ldots, s_k$ of the infinite computation $\sigma = s_0, \ldots, s_k, s_{k+1}, \ldots$.

Let II $\subseteq \Sigma^+$ be a finitary property. We define the following four properties of finite and infinite computations, $A(\Pi)$, $E(\Pi)$, $R(\Pi)$, $S(\Pi) \subseteq \Sigma^\infty$, by:

. $\sigma \in A(\Pi) \leftrightarrow \forall\sigma'(\sigma' \preceq a):\ \sigma' \in II$.

Obviously, $\sigma \in A($ *II) iff* every finite prefix of σ is in II.

We define the finitary and infinitary restrictions of $A(\Pi)$, by:

$$A_f(\Pi) = A(\Pi) \cap \Sigma^+ \quad \text{and} \quad A_\omega(\Pi) = A(\Pi) \cap \Sigma^\omega.$$

. $\sigma \in E(\Pi) \leftrightarrow \exists\sigma'(\sigma' \preceq \sigma): \sigma' \in II$.

Obviously, $\sigma \in E($ II) *iff* some finite prefix of σ is in Π.

We define:

$$E_f(\Pi) = E(\Pi) \cap \Sigma^+ \quad \text{and} \quad E_\omega(\Pi) = E(\Pi) \cap \Sigma^\omega.$$

- $\sigma \in R(\Pi) \leftrightarrow \forall\sigma'(\sigma' \preceq \mathbf{a}):\ \exists\sigma''(\sigma' \preceq \sigma'' \preceq \mathbf{a}):\ \sigma'' \in \mathrm{II}$.

Obviously, $\sigma \in R(\Pi)$ *iff*
either σ is finite and belongs to II,
or *infinitely many* finite prefices of σ are in II.

We define:

$$R_f(\Pi) = R(\Pi) \cap \Sigma^+ \quad \text{and} \quad R_\omega(\Pi) = R(\Pi) \cap \Sigma^\omega.$$

- $\sigma \in P(\Pi) \leftrightarrow \exists\sigma'(\sigma' \preceq$ a): $\forall\sigma''(\sigma' \preceq \sigma'' \preceq$ a): $\sigma'' \in$ II.

Obviously, $\sigma \in P(\Pi)$ *iff*
either σ is finite and belongs to Π,
or *all but finitely* many finite prefices of σ are in II.

We define:

$$P_f(\Pi) = P(\Pi) \cap \Sigma^+ \quad \text{and} \quad P_\omega(\Pi) = P(\Pi) \cap \Sigma^\omega.$$

We call $A_\omega(\Pi)$, $E_\omega(\Pi)$, $R_\omega(\Pi)$, and $P_\omega(\Pi)$ the *safety, guarantee, recurrence*, and *persistence* properties *induced* by II, respectively. We classify an infinitary property $\Pi \subseteq \Sigma^\omega$ as follows:

- Π is a *safety* property if $\Pi = A_\omega(\Pi')$ for some finitary II'.
- Π is a *guarantee* property if $\Pi = E_\omega(\Pi')$ for some finitary II'.
- II is a *recurrence* property if $\Pi = R_\omega(\Pi')$ for some finitary II'.
- Π is a *persistence* property if II $= P_\omega(\Pi')$ for some finitary II'.

We refer to these four classes of properties, denoted by $\mathcal{A}$, $\mathcal{E}$, $\mathcal{R}$, and $\mathcal{P}$, respectively, *as* the *basic* classes.

A property $\Pi \subseteq \Sigma^\omega$ is a *fairness* property if $\Pi = R_\omega(\Pi_1)$ **U** $P_\omega(\Pi_2)$ for some finitary Π_1 and Π_2. Let $\mathcal{F}$ denote the class of all fairness properties.

A property is called a *hyper-fairness* property if it is definable as a boolean combination of properties of the four basic classes. The class of all hyper-fairness properties is denoted by $\mathcal{H}$. Our proposal in this paper is to identify the intuitive notion of *liveness* with the class of hyper-fairness properties. The motivation and arguments in favor of this identification will be discussed in the next section.

We observe the following facts about the defined classes.

Fact 1 (Duality)

The classes $\mathcal{A}$ and $\mathcal{E}$ are dual under complementation, i.e., $\Pi \subseteq \Sigma^\omega$ is a *safety* property iff $\Sigma^\omega - \Pi$ is a *guarantee* property. Similarly, the classes $\mathcal{R}$ and $\mathcal{P}$ are dual. The class $\mathcal{H}$ is closed under complementation.

To show that $\mathcal{A}$ and $\mathcal{E}$ are complementary, we observe that for a finitary $\Pi \subseteq \Sigma^+$:

$$\Sigma^\omega - A_\omega(\Pi) = E_\omega(\Sigma^+ - \Pi), \text{ and}$$
$$\Sigma^\omega - E_\omega(\Pi) = A_\omega(\Sigma^+ - \Pi).$$

This is because

$$\begin{aligned}\sigma \in (\Sigma^\omega - A_\omega(\Pi)) &\leftrightarrow \sigma \notin A_\omega(\Pi)\\ &\leftrightarrow \neg(\forall\sigma'(\sigma' \preceq a)\colon \sigma' \in II))\\ &\leftrightarrow \exists\sigma'(\sigma' \preceq a)\colon \sigma' \notin \Pi\\ &\leftrightarrow \exists\sigma'(\sigma' \preceq a)\colon \sigma' \in (\Sigma^+ - II).\end{aligned}$$

Similarly:

$$\Sigma^\omega \text{ - R\&I) } = P_\omega(\Sigma^+ \text{ - II}), \text{ and}$$
$$\Sigma^\omega - P_\omega(\Pi) = R_\omega(\Sigma^+ - \Pi).$$

The class $\mathcal{H}$, being defined as properties obtained by a boolean combination, i.e., union, intersection, and complementation, of properties of the four basic classes, is certainly closed under one more application of the boolean operation of complement at ion.

Fact 2 (Closure)

The classes **d,** $\mathcal{E}, \mathcal{R}, \mathcal{P}$, and 7-L are closed under union and intersection. The class $\mathcal{F}$ is closed under union.

To show these closure properties, we need some operations on *finitary* properties. We use freely the boolean operations of union and intersection, and complement&ion with respect to Σ^+. We also use concatenation of properties:

$$\Pi_1 \cdot \Pi_2 = \{\sigma_1 \cdot \sigma_2 \mid \sigma_1 \in \Pi_1, \sigma_2 \in \Pi_2\}.$$

The special property Σ is the set of all singleton computations, i.e., computations consisting each of a single state.

An additional finitary operator is the *since* operator, modeled after the corresponding temporal operator. It is defined by:

(Π_1)S(Π_2) = {a $\in \Sigma^+ \mid \exists\sigma'(\sigma' \preceq$ a):
[(a' $\in \Pi_2)$ A $(\forall\sigma''(\sigma' \prec \sigma'' \preceq$ a): $\sigma'' \in$ II,)]}.

According to this definition $\sigma \in (\Pi_1)$S(Π_2) iff σ has a finite prefix σ' in II;! and all other prefices longer than σ' are in Π_1.

The closure properties under the positive boolean operations are shown as follows:

- For *safety,* they are justified by:

$$\begin{aligned} A_\omega(\Pi_1) \cap A_\omega(\Pi_2) &= A_\omega(\Pi_1 \cap \Pi_2), \\ A_\omega(\Pi_1) \cup A_\omega(\Pi_2) &= A_\omega(A_f(\Pi_1) \cup A_f(\Pi_2)). \end{aligned}$$

To support the last equality, we show inclusion in both directions.

Assume that $\sigma \in A_\omega(\Pi_1)$. This means that every finite prefix $\sigma' \preceq \sigma$ is in Π_1. Take any finite prefix $\sigma'' \preceq \sigma$. Obviously, any finite prefix of σ'' is also a finite prefix of σ and hence is in Π_1. It follows that $\sigma'' \in A(\Pi_1)$. Since σ'' is finite, actually $\sigma'' \in A(\Pi_1) \cap \Sigma^+ = A_f(\Pi_1)$. Clearly $\sigma'' \in A_f(\Pi_1)$ **U** $A_f(\Pi_2)$. Hence, any finite prefix of σ is in $A_f(\Pi_1)$ **U** $A_f(\Pi_2)$. We conclude that $\sigma \in A_\omega(A_f(\Pi_1)$ **U** $A_f(\Pi_2))$, and therefore:

$$A_\omega(\Pi_1) \subseteq A_\omega(A_f(\Pi_1) \cup A_f(\Pi_2)).$$

By symmetry, also $A_\omega(\Pi_2)$ is contained in the same right hand side, and we conclude

$$A_\omega(\Pi_1) \cup A_\omega(\Pi_2) \subseteq A_\omega(A_f(\Pi_1) \cup A_f(\Pi_2)).$$

To show inclusion in the other direction, assume that $\sigma \notin A_\omega(\Pi_1) \cup A_\omega(\Pi_2)$. Then σ must have a finite prefix $\sigma_1 \notin \Pi_1$ and another finite prefix $\sigma_2 \notin \Pi_2$. Without loss of generality assume $\sigma_1 \preceq \sigma_2$. It follows that $\sigma_2 \notin A_f(\Pi_2)$, and since it has a prefix $\sigma_1 \notin \Pi_1$, also $\sigma_2 \notin A_f(\Pi_1)$. Hence $\sigma_2 \notin A_f(\Pi_1) \cup A_f(\Pi_2)$, and therefore $\sigma \notin A_\omega(A_f(\Pi_1) \cup A_f(\Pi_2))$.

- For *guarantee,* we use the previously established duality with *safety* to claim:

$$\begin{aligned} E_\omega(\Pi_1) \cup E_\omega(\Pi_2) &= E_\omega(\Pi_1 \cup \Pi_2), \\ E_\omega(\Pi_1) \cap E_\omega(\Pi_2) &= E_\omega(E_f(\Pi_1) \cap E_f(\Pi_2)). \end{aligned}$$

- For *recurrence*, we claim:

$$R_\omega(\Pi_1) \cup R_\omega(\Pi_2) = R_\omega(\Pi_1 \cup \Pi_2).$$

Obviously σ contains either infinitely many Π_1-prefices or infinitely many Π_2-prefices *iff* it contains infinitely many $(\Pi_1 \cup \Pi_2)$-prefices.

Closure under intersection is given by the equality:

$$R_\omega(\Pi_1) \cap R_\omega(\Pi_2) = R_\omega(\Pi_1 \cap [(\Sigma^+ - \Pi_1)\mathbf{S}(\Pi_2)] \cdot \Sigma).$$

We observe t hat $\sigma \in (\Pi_1 \cap [(\Sigma^+ - \Pi_1)\mathbf{S}(\Pi_2)] \cdot \Sigma)$ *iff* $\sigma \in \Pi_1$, σ has a prefix $\sigma' \prec \sigma$ such that $\sigma' \in \Pi_2$, and all other prefices, longer than σ' and shorter than σ, are not in Π_1. This characterizes a finite computation σ in Π_1, such that its longest proper prefix which belongs to $\Pi_1 \cup \Pi_2$ belongs in fact to II;!.

Obviously, σ has infinitely many Π_1-prefices as well as infinitely many Π_2-prefices *iff* it has infinitely many Π_1-prefices whose longest proper $(\Pi_1 \cup \Pi_2)$-prefix is a Π_2-prefix.

- For *persistence,* we use the duality with the *recurrence* class. This yields:

$$P_\omega(\Pi_1) \cap P_\omega(\Pi_2) = P_\omega(\Pi_1 \cap \Pi_2),$$

$$P_\omega(\Pi_1) \cup P_\omega(\Pi_2) = P_\omega(\Pi_1 \cup [(\Pi_2)\mathbf{S}(\Sigma^+ - \Pi_1)] \cdot \Sigma).$$

- The class of *fairness* properties is closed under unions. To see this, we observe that

$$[R_\omega(\Pi_1) \cup P_\omega(\Pi_2)] \cup [R_\omega(\Pi'_1) \cup P_\omega(\Pi'_2)] = R_\omega(\Pi''_1) \cup P_\omega(\Pi''_2),$$

where

$R_\omega(\Pi''_1) = R_\omega(\Pi_1) \cup R_\omega(\Pi'_1)$, guaranteed by the closure properties of $\mathcal{R}$, and

$P_\omega(\Pi''_2) = P_\omega(\Pi_2) \cup P_\omega(\Pi'_2)$, guaranteed by the closure properties of $\mathcal{P}$.

- The closure of $\mathcal{H}$ under any boolean operations is obvious.

Fact 3 (Inclusion)

The classes of properties are related by the inclusion relations depicted in the diagram of Fig. 1. The edges in the diagram represent strict inclusions.

To show the inclusions $\mathcal{A} \cup \mathcal{E} \subseteq \mathcal{R}$ and $\mathcal{A} \cup \mathcal{E} \subseteq \mathcal{P}$, we observe that:

$$\begin{aligned} A_\omega(\Pi) &= R_\omega(A_f(\Pi)) = P_\omega(A_f(\Pi)), \\ E_\omega(\Pi) &= R_\omega(E_f(\Pi)) = P_\omega(E_f(\Pi)). \end{aligned}$$

For example, an infinite $\sigma \in R_\omega(E_f(\Pi))$ *iff* σ has infinitely many prefices $\sigma' \in E_f(\Pi)$, i.e., prefices σ' containing a prefix $\sigma'' \preceq \sigma'$ such that $\sigma'' \in \Pi$. This is true *iff* σ has some prefix $\sigma'' \prec \sigma$ such that $\sigma'' \in \Pi$. Hence $R_w(E_f(\Pi)) = E_w(\Pi)$.

The other inclusions are equally easy to show. The strictness of the inclusions between the classes below $\mathcal{H}$ is also straightforward to show.

Corollary (Normal Form)

Any *hyper-fairness* property is expressible as the intersection of several *fairness* properties.

Let $\Pi \subseteq \Sigma^\omega$ be a hyper-fairness property. By definition it can be expressed as a. boolean combination of properties in the classes $\mathcal{A}$, $\mathcal{E}$, $\mathcal{R}$ and $\mathcal{P}$. We perform the following transformations on the boolean expression:

First we push all the complementations inside. By the closure properties described in Fact 1, all the complementations can be reduced to operations on finitary properties.

We are left with a positive boolean combination of properties of the basic four classes. Use the inclusion relations of Fact 3 to reexpress all $\mathcal{A}$ and $\mathcal{E}$ properties as $\mathcal{R}$ and $\mathcal{P}$ properties.

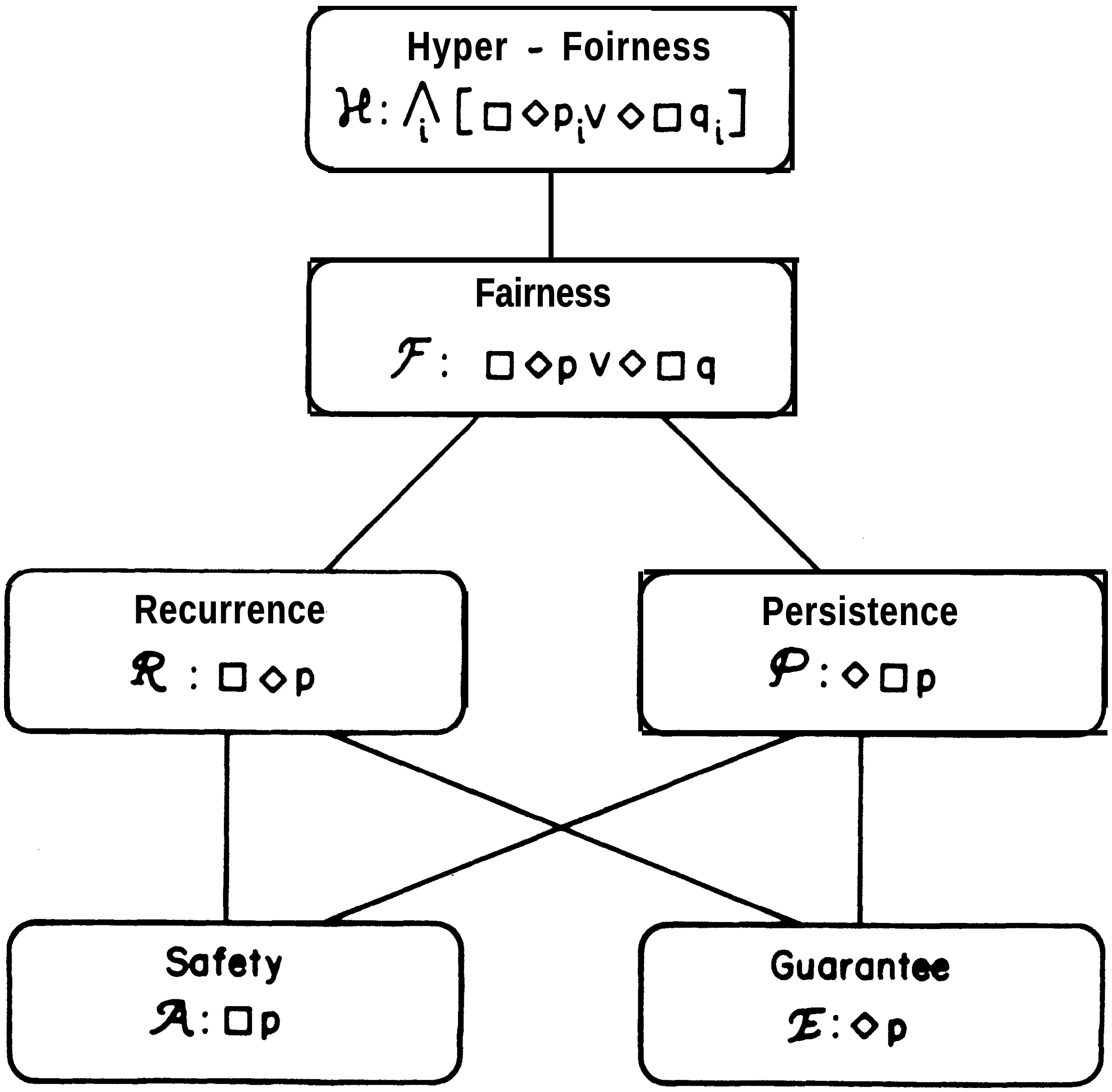
Hyper - Foirness
$\mathcal{H}: \bigwedge_i [\Box\Diamond p_i \vee \Diamond\Box q_i]$
Fairness
$\mathcal{F}: \Box\Diamond p \vee \Diamond\Box q$
Recurrence
$\mathcal{R}: \Box\Diamond p$
Persistence
$\mathcal{P}: \Diamond\Box p$
Safety
$\mathcal{A}: \Box p$
Guarantee
$\mathcal{E}: \Diamond p$

Fig. 1 . Inclusion Relat ions between the Classes.

Expand the resulting expression by distributivity to obtain a conjunctive normal form, i.e., an intersection of unions of $\mathcal{R}$ and $\mathcal{P}$ properties. By the closure properties of Fact 2, each union can be collapsed to a union of a single $\mathcal{R}$ property and a single $\mathcal{P}$ property. It follows that the hyper-fairness property can be expressed as:

$$\bigcap_{i=1}^{k} \left[R_\omega(\Pi_i^1) \cup P_\omega(\Pi_i^2)\right],$$

which is of course an intersection of fairness properties.

We may in fact consider a complete hierarchy of hyper-fairness properties $\mathcal{F}_1$ c $\mathcal{F}_2$ c ..., where $\mathcal{F}_k$ for $k \geq 1$ is defined as the class of all properties that can be expressed as the intersection of k fairness properties. As will be discussed later, it can be shown that the sequence $\mathcal{F}_1 \subset \mathcal{F}_2 \subset \ldots$ forms a. strict hierarchy. Obviously $\mathcal{H} = \bigcup_{k \geq 1} \mathcal{F}_k$, and the strictness of the hierarchy implies the strict inclusion of $\mathcal{F} = \mathcal{F}_1$ in $\mathcal{H}$.

We refer to the layers $\mathcal{F}_n$, $n > 1$, as the higher levels of the hierarchy. The hierarchy *we* study here is a. *conjunctive* one, meaning that the outermost operator is an intersection or conjunction. The hierarchy studied in [Lan], [Wag] and [Kam] is *disjunctive.* Its k-th layer is given as:

$$\bigcup_{i=1}^{k} \left[R_\omega(\Pi_i^1) \cap P_\omega(\Pi_i^2)\right].$$

Obviously? these two versions are dual, and propertics of one can be mapped to properties of the other by complementation.

Comparison with Other Semantic Definitions

In this section we would like to compare our definition with some alternative characterizations of the safety-liveness classification, in particular to that of [AS1] and [S].

A safety property is characterized in [AS1] as a property II $\subseteq \Sigma^\omega$ such that

$$\sigma \in \Pi \quad \leftrightarrow \quad \forall \sigma'(\sigma' \preceq a) : \exists \sigma''(\sigma'' \in \Sigma^\omega) \colon \sigma' \cdot \sigma'' \in \Pi. \qquad (*)$$

To see that this characterization precisely matches ours, we observe that the expression

$$\exists \sigma''(\sigma'' \in \Sigma^\omega) \colon \sigma' \cdot \sigma'' \in \Pi$$

can be rewritten as $\sigma' \in Pref\,(\Pi)$ for an infinitary Π. Hence, the characterization is equivalent to:

$$\sigma \in \Pi \quad \leftrightarrow \quad \forall\sigma'(\sigma' \preceq \sigma)\colon \sigma' \in Pref\,(\Pi) \quad \leftrightarrow \quad \sigma \in A_\omega(Pref\,(\Pi)).$$

This is the same as:

$$\Pi = A_\omega(Pref\,(\Pi)).$$

It follows that any property satisfying $(*)$ can be expressed as $A_\omega(\Pi')$, where Π' happens to be $Pref\,(\Pi)$. It is not difficult to see that, if $\Pi = A_\omega(\Pi')$ for an arbitrary Π', then in fact Π can also be expressed as $A_\omega(Pref\,(\Pi))$.

Unfortunately, safety is where the agreement, between the various definitions stops.

The definition in [AS1] characterizes a liveness property as an infinitary property $\Pi \subseteq \Sigma^\omega$, such that

$$\forall\sigma(\sigma \in \Sigma^+)\colon \exists\sigma'(\sigma' \in \Sigma^\omega)\colon \sigma \cdot \sigma' \in \Pi.$$

That is, every finite computation can be extended to an infinite one which belongs to Π. Our characterization of hyperfairness (liveness) is quite different from this definition, and can almost be described as orthogonal to the approach taken in [AS1].

The differences are to a large extent differences in the motivation for wishing to distinguish between safety and liveness. The definition of [AS1] is an attempt to formalize the intuitive description of Lamport in [L] of liveness as a property stating that

"Something good will eventually happen,"

as dual to safety, which is described as

"Nothing bad ever happens."

There are two main points in which the current paper wishes to expand this view.

The first is that the "something good will eventually happen" is only a partial characterization of liveness, adequate perhaps for the case of finite computations, where liveness usually deals with termination, an event that is expected to happen only once. This is characterized by our class $\mathcal{E}$. However, when considering infinite computations, there are at least two more basic forms of liveness, the one states that "something good will happen infinitely often," and the other states that "from a certain point on, only good things will happen." The approach we present in this paper is that these three basic forms of liveness-like utterances should be recognized separately.

Another point is that our interpretation of "something good" is that of a finite-prefix property, which refers not only to the current state but in general to the full preceding history. This leads to the usage of past formulae as expressing properties of finite prefices, and causes another division point with the approach in [AS1].

In this paper we refer to the class of liveness properties defined in [AS1] as pure Ziveness. This is based on the interpretation that the definition of [AS1] tries to purify the concept of liveness from any trace of safety, while our definition allows safety constraints as part of a liveness property.

It is possible to "purify" each of our classes of any taint of safety in the spirit of [AS1].

We define a property Π to be *a pure* $\mathcal{E}$, $\mathcal{R}$, $\mathcal{P}$, $\mathcal{F}$, or $\mathcal{H}$ property, if it is a pure liveness property and also belongs to $\mathcal{E}$, $\mathcal{R}$, $\mathcal{P}$, $\mathcal{F}$, or $\mathcal{H}$, respectively.

Given a property II in any of the classes $\mathcal{E}$, $\mathcal{R}$, $\mathcal{P}$, or $\mathcal{H}$, we can define

$$\text{Pure (II)} = II \cup E_\omega(\Sigma^+ - Pref(\Pi)).$$

Clearly, $\sigma \in$ *Pure* (II) *iff* either $\sigma \in$ II or σ contains a prefix $\sigma' \prec \sigma$ that *cannot* be extended to a computation in II. Take any finite computation σ_0. Either it can be extended to a computation in II, or it belongs to $\Sigma^+ - Pref(\Pi)$, in which case any infinite extension of it will belong to $E_\omega(\Sigma^+ - Pref(\Pi))$. This shows that $Pure(\Pi)$ is a pure liveness property. Due to the closure properties of the classes, I, $\mathcal{R}$, $\mathcal{P}$, and $\mathcal{H}$, $Pure(\Pi)$ belongs to the same class to which II belongs. It follows that $Pure(\Pi)$ is, respectively, a pure $\mathcal{E}$, $\mathcal{R}$, $\mathcal{P}$, $\mathcal{F}$, or $\mathcal{H}$ property.

Note that the purification of a property usually enlasges it. This is because it removes from the constraints defining the property all those which have the character of safety, and consequently admits additional computations.

One of the results of [AS1] is that each X-property is the intersection of a pure liveness property with a safety property. This result can now be extended to state that, for $\alpha \in$ {I, $\mathcal{R}$, $\mathcal{P}$, $\mathcal{F}$, $\mathcal{H}$}, each o-property can be represented as the intersection of a. pure α-property and a safety property. This is due to the equality:

$$II = Pure(\Pi) \cap \mathcal{A}_\omega\left(Pref(\Pi)\right).$$

Topological Characterization

It is possible to assign a topological identification to the classes of properties considered above. A natural topology can be introduced into the space Σ^ω by defining the distance between two computations σ and $\sigma' \in \Sigma^\omega$ to be:

$$\rho(\sigma, \sigma') = 2^{-k},$$

where k is the minimal index i such that $\sigma[i] \neq \sigma'[i]$.

With this topology we can establish the following correspondence between our classification and the first levels of the Borel hierarchy:

($\mathcal{A} = F$) Π is a *safety* property iff it is a *closed* set.

($\mathcal{E} = G$) Π is a *guarantee* property iff it is an *open* set.

($\mathcal{R} = G_\delta$) Π is a recurrence property iff it is a G_δ set.

($\mathcal{P} = F_\sigma$) Π is a *persistence* property iff it is an F_σ set.

In the above we have denoted by F the family of all closed sets, by G the family of all open sets, by G_δ all sets obtainable as a. countable intersection of open sets. and by F_σ all sets obtainable as a countable union of closed sets.

All hyper-fairness properties are contained in both $G_{\delta\sigma}$ and $F_{\sigma\delta}$, i.e., belong to $G_{\delta\sigma} \cap F_{\sigma\delta}$. Recall that $G_{\delta\sigma}$ are the sets obtainable as a countable union of G_δ sets, and $F_{\sigma\delta}$ are the sets obtainable as a countable intersection of F_σ sets.

2. Expressiveness in Temporal Logic

Next, we restrict our attention to infinitary properties that can be expressed in temporal logic. We use the version of temporal logic defined in [LPZ]. It includes, among others, the future operators $\square$ ("henceforth") and $\diamondsuit$ ("eventually"), and the past operators $\ominus$ ("previously"), $\boxminus$ ("till-now"), $\diamondminus$ ("once")! and $\mathcal{S}$ ("since").

We define the truth of temporal formulae at position $i \geq 0$, in an infinite computation $\sigma: s_0, s_1, \ldots$, in the following way:

$(\sigma, i) \models p \quad \leftrightarrow s_i \models p$, for a state (non-temporal) formula p

$(\sigma, i) \models p \vee q \leftrightarrow (\sigma, i) \models p$ or $(\sigma, i) \models q$

$(\sigma, i) \models \neg p \quad \leftrightarrow$ not $(\sigma, i) \models p$

$(\sigma, i) \models \Diamond p \leftrightarrow \exists k (k \geq i) \colon (\sigma, k) \models p$

$(\sigma, i) \models \ominus p \leftrightarrow (i > 0)$ and $(\sigma, i - 1) \models p$

$$(\sigma, i) \models p \mathcal{S} q \leftrightarrow \exists k (k \leq i) \colon \begin{bmatrix} (\sigma, k) \models q \wedge \\ \forall j (i < j \leq k) \colon (\sigma, j) \models p \end{bmatrix}$$

We define:

$$\begin{aligned} \Box p &= \neg \Diamond (\neg p), \\ \diamondminus p &= true\ \mathcal{S}\ p, \\ \boxminus p &= \neg \diamondminus (\neg p). \end{aligned}$$

We abbreviate $\Box (p \rightarrow q)$ to $p \Rightarrow q$, saying that p *entails* q (we use $\rightarrow$ for implication). We are not very specific about the language in which state-formulae, also called assertions, are expressed. An example is a first-order language over some theory such as the integers. A computation σ satisfies a temporal formula p, denoted by $\sigma \models p$, if $(a, 0) \models p$. A formula p specifies a property $II(p)$ given by:

$$\Pi(p) = \{ a \in \Sigma^{\omega} \mid \sigma \models p \}.$$

Two formulae p and q ase defined to be equivalent,, $p \approx q$, if $\Pi(p) = II(q)$. Note that when we stake that $p \approx q$, we mean that $p \equiv q$ in the *first* position of every computation.

Below, we present for each class of properties a syntactic characterization of the formulae that specify properties in that class, examples of some formulae of alternative forms that also specify properties in that class, and some comments about boolean closures of the class.

- *Safety*

A formula of the form $\Box p$ for some past formula p is called a. *safety* formula,. Obviously, every safety formula specifies a safety property.

Conversely, every safety property which is specifiable in temporal logic, is specifiable by a safety formula,. This means that every infinitary property II that is expressible, on one hand, as $A_{\omega}(\Pi')$ for some finitary II', and is specifiable, on the other hand, by *some* temporal formula, is specifiable in fact by a. safety formula.

To see this, we observe that for every temporal formula φ, there exists an effectively derivable past formula $\mathit{prefix}(\varphi)$ such that for each $\sigma \in \Sigma^{\omega}$ and $k \geq 0$

$$(\sigma, k) \models \mathit{prefix}(\varphi) \leftrightarrow \exists\sigma'(\sigma' \in \Sigma^{\infty})\colon \sigma[0 \,..\, k] \cdot \sigma' \models \varphi.$$

This means that $\mathit{prefix}(\varphi)$characterizes all the finite computations that can be extended to computations satisfying φ. Then, if φ specifies a safety property, it can be shown that $\varphi \approx \Box$ *prefix(y).*

Examples of properties specified by safety formulae are partial correctness, mutual exclusion, absence of deadlock, etc. The closure of safety formulae under conjunction and disjunction is based on the following equivalences:

$$(\Box p \wedge \Box q) \approx \Box(p \wedge q),$$
$$(\Box p \vee \Box q) \approx \Box(\boxminus p \vee \boxminus q).$$

Note the analogy with the corresponding proof of closure for the semantic view.

- *Guarantee*

A formula of the form $\Diamond p$ for some past formula p is called a *guarantee* formula. Obviously every guarantee formula specifies a guarantee property.

Conversely, every guasantee property which is specifiable in temporal logic can be specified by a guarantee formula. To see this we observe that, if φ specifies a guarantee property, then $\varphi \approx \Diamond(\neg\mathit{prefix}(\neg\varphi))$.

Examples of properties specifiable by guarantee formulae are total correctness, termination, and guarantee of a. goal that has to be reached **once.** The closure of guarantee formulae under conjunction and disjunction is ensured by the equivalences:

$$(\Diamond p \vee \Diamond q) \approx \Diamond(p \vee q),$$
$$(\Diamond p \wedge \Diamond q) \approx \Diamond(\diamondminus p \wedge \diamondminus q).$$

- *Recurrence*

A formula. of the form $\Box\Diamond p$ for some past formula p is called a *recurrence* formula. A recurrence formula. obviously specifies a recurrence property.

Conversely, every recurrence property which is specifiable in temporal logic can be specified by a recurrence formula. This fact will be shown later.

An alternative useful form for recurrence properties is the entailment $p \Rightarrow \Diamond q$ or, equivalently, $\Box$ $(p \rightarrow \Diamond q)$. To see that this formula, specifies a recurrence property we observe:

$$(p \Rightarrow \Diamond q) \approx \Box \quad O(\boxminus\neg p \vee [(\neg p)\mathcal{S} q]).$$

The formula on the right states the existence of infinitely many states such that the last observed p was followed by (or coincided with) q. Recurrence formulae can specify all the properties specifiable by safety formulae. This is due to the equivalence:

$$\Box p \approx \Box\Diamond(\boxminus p).$$

They can also specify all the properties specified by guarantee formulae:

$$\Diamond p \approx \Box\Diamond(\diamondminus p).$$

Examples of properties specifiable by recurrence formulae are accessibility, lack of individual starvation, responsiveness to requests, etc. Recurrence formulae can also express weak fairness requirements. A weak fairness requirement for a transition τ in a program is that, if τ is continuously enabled beyond some point, it will eventually be taken. This can be expressed by:

$$(\Diamond\Box En(\tau) \Rightarrow \Diamond taken(\tau)) \approx \Box\Diamond(\neg En(\tau) \vee taken(\tau)).$$

The closure of recurrence formulae under conjunction and disjunction is ensured by the equivalences:

$$(\Box\Diamond p \vee \Box\Diamond q) \approx \Box\Diamond(p \vee q),$$
$$(\Box\Diamond p \wedge \Box\Diamond q) \approx \Box\Diamond[p \wedge (q \vee \ominus[(\neg p)\,\mathcal{S}\,q])].$$

• *Persistence*

A formula, of the form $\Diamond\Box p$ for some past formula p is called a. *persistence* formula.. Persistence formulae obviously specify persistence properties.

Conversely, every persistence property which is specifiable in temporal logic can be specified by a persistence formula. This follows by duality from the corresponding result for recurrence formulae.

Similarly to recurrence, persistence formulae can specify all the properties specifiable by safety and guaraatee formulae. This is supported by:

$$\Box p \approx \Diamond\Box(\boxminus p),$$
$$\Diamond p \approx \Diamond\Box(\diamondminus p).$$

The closure of persistence formulae under conjunction and disjunction can be obtained by duality from the closure properties of recurrence formulae.

- *Fairness*

A formulaof the form $\square\Diamond p \vee \Diamond\square q$ for some past formulae p and q is called *a fairness* formula. Obviously, a fairness formula specifies a fairness property.

Conversely, every fairness property which is specifiable in temporal logic can be specified by a fairness formula. This will be shown later.

It is easy to see that fairness formulae generalize both recurrence and persistence formulae. An alternative form for fairness formulae is: $\square\Diamond p \rightarrow \square\Diamond q$. In this form they are useful for specifying *strong* fairness requirements, such as $\square\Diamond En(\tau) \rightarrow \square\Diamond taken(\tau)$, which states that a transition which is enabled infinitely many times must be taken infinitely many times. Fairness formulae can also describe systems whose response is guaranteed only if there are infinitely many requests for this response. An example of such a system is an eventually reliable channel .

Fairness formulae are closed under disjunction but not under conjunction. A conjunction of fairness formulae leads to the most general normal form of temporal formulae:

$$\bigwedge_{i=1}^{n} [\square\Diamond p_i \vee \Diamond\square q_i],$$

which are identified as general *hyper-fairness* formulae.

We can summarize the relation of the property hierarchy to the formula, hierarchy by the following proposition.

Proposition

A property Π, that is specifiable by a temporal formula, is an α-property iff it is specifiable by an α-formula, where $\alpha \in$ *{safety, guarantee, recurrence. persistence, fairness)*.

The fact that every a-formula, specifies an a-property is straightforward. The other direction has been proved for the *safety* and *guarantee* cases. For the other cases we have to rely on a similar proof for automata,, which we discuss next.

3. Predicate Automata

An alternative formalism for specifying temporal properties is given by finite-state predicate automata (see [AS2], [MP]). In the version we consider here, a predicate automaton $\mathcal{M}$ consists of the following components:

Q – A finite set of automaton-states.

$q_0 \in Q$ – An *initial* automaton-state.

$T = \{t(q_i, q_j) | q_i, q_j \in Q\}$ – A set of *transition conditions*. For each $q_i, q_j \in$ Q, $t(q_i, q_j)$ is a state formula specifying the computation-states under which the automaton may proceed from q_i to q_j. It is assumed that $t(q, q_0)$ = *false* for every $q \in$ Q. We also assume that each $t(q_i, q_j)$ is either syntactically identical to the constant *false,* or holds over some computation-state s.

$R \subseteq Q$ – A set of *recurrent* automaton-states.

$P \subseteq Q$ – A set of *persistent* automaton-states.

Let

$$\sigma: s_0, s_1, \ldots \in \Sigma^\omega$$

be an infinite computation. Computations are fed as input to the automaton, which either accepts or rejects them. An infinite sequence of automaton-states

$$r: q_0, q_1, \ldots \in Q^\omega$$

is called a *run* of $\mathcal{M}$ over σ if:

1. q_0 is the initial state of $\mathcal{M}$
2. for every i > 0, $s_{i-1} \models t(q_{i-1}, q_i)$.

Note that the automaton always starts at q_0, and s_0 causes it to move from q_0 to q_1.

We define the infinity set of r, $Inf(r)$, to be the set of automaton-states that, occur infinitely many times in r.

A run r is defined to be *accepting* if either $Inf(r) \cap R \neq \emptyset$ or $Inf(r) \subseteq P$. The automaton $\mathcal{M}$ *accepts* the computation σ if there *exists* a run of M over σ which is accepting. This definition of acceptance has been introduced by Streett ([St]).

An alternative definition, given in [MP], is that *all* runs of M over σ are accepting.

The automaton $\mathcal{M}$ is called *complete* if, for each $q \in Q$,

$$\left(\bigvee_{q' \in Q} t(q, q') \right) \equiv true.$$

It is called *deterministic* if, for every q and $q' \neq q''$, $t(q, q') \rightarrow \neg t(q, q'')$, that is, we cannot have both $t(q, q')$ and $t(q, q'')$.

In this paper we restrict our attention to complete deterministic automata. In deterministic automata there is exactly one run r corresponding to each input computation σ, and hence the definition of acceptance in [MP] coincides with the one used here.

Let $G = R \cup P$ and $B = Q - G$. We refer to G and B as the "good" and "bad" sets of states, respectively. We define the following classes of automata by introducing restrictions on their transition conditions and accepting states:

- A *safety* automaton is such that, for every $q \in B$, $q' \in G$, $t(q, q') = false$. That is, it cannot move from a bad state $q \in B$ to a good state $q' \in G$.
- A *guarantee* automaton is such that, for every $q \in G$, $q' \in B$, $t(q,q') = false$.
- A *recurrence* automaton is such that $P = \emptyset$.
- A *persistence* automaton is such that $R = \emptyset$.
- A *fairness* automaton is an unrestricted predicate automaton.

We define the property specified by an automaton M, $\Pi_{\mathcal{M}}$, *as* the set of all infinite computations that ase accepted by M.

In order to attain expressive power comparable to (and even exceeding, see [W]) that of temporal logic we have to consider a more general type of automaton.

We define a. *hyper-fairness* automaton (*liveness* automaton) to be a structure

$$\mathcal{M} = \langle Q, q_0, T, L \rangle,$$

where Q, q_0, and T ase as defined above, and L is a finite set of pairs of acceptance sets:

$$L = \{(R_1, P_1), \ldots, (R_k, P_k)\}.$$

A run r of a liveness-automaton is accepting if, for *each* i = 1,. . . k, either $Inf(r) \cap R_i \neq \emptyset$ or $Inf(r) \subseteq P_i$. The notions of computations accepted by such an automaton and the properties specified by it are similar to the simpler case.

Obviously, all the preceding types of automata are special cases of hyper-fairness automata with $k = 1$. The hyper-fairness automaton is almost identical to the automaton studied by Streett in [St].

Proposition

A property Π, that is specifiable by automata, is an o-property iff it is specifiable by an α-automaton, where $\alpha \in$ {*safety, guarantee, recurrence, persistence, fairness* }.

For the first four types, this proposition has been proved in [Lan], with some minor differences in the definitions of safety and hyper-fairness automata. The case of fairness, and in fact the complete hierarchy above, has been solved in [Wag].

For completeness, we include below our version of a proof of the proposition, which for most of the cases is straightforward.

Proof

It is simple to show that an a-automaton specifies an a-property. Let $\mathcal{M}$ be an a-automaton. Since $\mathcal{M}$ is deterministic and complete, there is, for each finite computation $\sigma \in \Sigma^+$, a unique state q, denoted by $\delta(q_0, \sigma)$, such that the run of $\mathcal{M}$ on σ terminates (a is finite) at q.

Define II, = {a $\in \Sigma^+ \mid \delta(q_0, \sigma) = q\}$ for each $q \in Q$.

Obviously, an infinite σ is accepted by $\mathcal{M}$ *iff* its corresponding run r either visits infinitely many times states in R, or is constrained from a certain point to visit only P-states. This means that either σ contains infinitely many prefices in II, for $q \in R$, or that all but finitely many prefices of σ are each in some II, for $q \in P$. It follows that

$$\Pi_{\mathcal{M}} = R_\omega(\bigcup_{q \in R} \Pi_q) \cup P_\omega(\bigcup_{q \in P} \Pi_q).$$

Consequently, every property specifiable by a single automaton is a fairness property. However, as we will show for the special cases of an α-automaton, this expression can be further simplified.

- For a safety automaton, it is clear that no finite prefix of an acceptable computation caa be in $\Pi_B = \bigcup_{q \in B}$ II,. This is because, once a run visits a bad state $q \in B$. it can never return to a good state. Hence for safety automata we also have

$$\Pi_{\mathcal{M}} = A_\omega(\bigcup_{q \in G} \Pi_q),$$

which establishes $\Pi_{\mathcal{M}}$ as a safety property.

- For a guarantee automaton, once a run visits a good state it can never visit a bad state. It follows that

$$\Pi_{\mathcal{M}} = E_\omega(\bigcup_{q \in G} \Pi_q),$$

which shows that II$_{\mathcal{M}}$ is a guarantee property.

- For a recurrence automaton, we are given that $P = \emptyset$, and therefore

$$\Pi_{\mathcal{M}} = R_\omega(\bigcup_{q \in R} \Pi_q).$$

- For a persistence automaton, we are given that $R = \emptyset$, and therefore

$$\Pi_{\mathcal{M}} = P_\omega(\bigcup_{q \in P} \Pi_q).$$

Consider now the other direction of the proposition. It states that an α-property specifiable by automata can be specified by an a-automaton. Assume that an α-property II is specifiable by automata. Thus, there exists a liveness automaton

$$\mathcal{M} = \langle Q, q_0, T, L \rangle, \qquad L = \{(R_1, P_1), \ldots, (R_k, P_k)\}$$

specifying *II*.

Let δ: Q $\times \Sigma^+ \rightarrow$ Q be the function, based on T, that, for each state $q \in Q$ and finite computation $\sigma \in \Sigma^+$, yields the state $\delta(q$, a) $\in$ Q reached by the automaton starting at q after reading the computation σ.

- Consider first the case, that Π is a safety property, and hence satisfies II $=$ $A_\omega(\mathit{Pref}(\Pi))$.

We construct an automaton:

$$\mathcal{M}' = \langle Q, q_0, T', G, G \rangle,$$

where Q and q_0 are as before. G and B ase defined by:

$$G = \{q_0\} \cup \{q \in Q \mid \delta(q_0, \sigma) = q \text{ for some } \sigma \in \mathit{Pref}(\Pi)\},$$
$$B = Q - G.$$

The transition conditions $T' = \{t'(q, q') | q, q' \in Q\}$ are given by:

$$t'(q,q') = \begin{cases} \mathit{true} & q \in B, q' = q \\ \mathit{false} & q \in B, q' \neq q \\ t(q,q') & 4 \quad 4 \quad B. \end{cases}$$

We claim that, for a finite computation $\sigma \in \Sigma^+$,

$$\sigma \in Pref(\Pi) \quad \leftrightarrow \quad \delta(q_0, \sigma) \in G.$$

By the construction of G, if $\sigma \in Pref(\Pi)$, then $\delta(q_0, \sigma) \in G$.

Assume that $\sigma \notin Pref(\Pi)$. This means that σ cannot be a prefix of a computation in Π. Let $\delta(q_0, \sigma) = q$. We would like to show that $q \notin G$.

Assume to the contrary that $q \in G$. This can only be caused by another finite computation $\sigma' \in Pref(\Pi)$ such that also $\delta(q_0, \sigma') = q$. If $\sigma' \in Pref(\Pi)$, there must exist an extension $\sigma'' \in \Sigma^\omega$, such that $\sigma' . \sigma'' \in \Pi$ and hence is accepted by $\mathcal{M}$. Consider the mixed computation $\sigma . \sigma'' \in \Sigma^\omega$. Let r be the run of (Q, q_0, T) over $\sigma . \sigma''$, and r' the run of (Q, q_0, T) over $\sigma' . \sigma''$. Since $\delta(q_0, \sigma) = \delta(q_0, \sigma') = q$, these runs coincide after a finite segment. It follows that $Inf(r) = Inf(r')$, and hence $\sigma . \sigma''$ should be accepted by M. This contradicts our assumption that $\sigma \notin Pref(\Pi)$. Hence our claim is established.

It is now easy to show that $\sigma \in \Sigma^\omega$ is accepted by $\mathcal{M}'$ iff $\sigma \in \Pi$.

Denote by δ' the transition function based on T'. Assume that σ is accepted by $\mathcal{M}'$, and let r be its corresponding run. To be accepting, r must go infinitely many times through G-states. By the way we defined T', this means that M' only visits G-states. Since T and T' are identical as long as we only visit G-states, this means that, for every $\sigma' \prec \sigma$, $\delta(q_0, \sigma') = \delta'(q_0, \sigma') \in G$. It follows that every $\sigma' \prec \sigma$ is in $Pref(\Pi)$, and since Π is a safety property, that $\sigma \in \Pi$.

In the other direction, assume that σ is rejected by M'. This implies the existence of a *minimal* $\sigma' \prec \sigma$ such that $\delta'(q_0, \sigma') \notin G$. Since σ' is minimal, the run caused by σ' visits only G-states except the last. It follows that $\delta'(q_0, \sigma') = \delta(q_0, \sigma') \notin G$, and hence $\sigma' \notin Pref(\Pi)$. Thus, σ' cannot, be the prefix of a computation in Π, and therefore $\sigma \notin \Pi$.

■ Consider the case that Π is a *guarantee* property.

In that case, we have that $\Pi = E_\omega(\Pi')$ for some finitary property Π'. We define the sets G and B, as follows:

$$G = \{q \mid \delta(q_0, \sigma) = q \text{ for some } \sigma \in \Pi'\},$$
$$B = Q - G.$$

Construct the automaton:

$$\mathcal{M}' = \langle Q, q_0, T', G, G \rangle,$$

where T'is given by:

$$t'(q,q') = \begin{cases} true & q \in G, q' = q \\ false & q \in G, q' \neq q \\ t(q,q') & q \notin G. \end{cases}$$

We show that $\sigma \in \Sigma^\omega$ is accepted by $\mathcal{M}'$ iff $\sigma \in$ Il.

Assume that σ is accepted by M'. Then there exists some prefix $\sigma_1 \prec \sigma$ which causes M' to visit a state in G for the first time while reading σ. Let $q = \delta'(q_0, \sigma_1)$. Since q is the first visit to a G-state, it follows that the behavior of M' on σ_1 is identical to that of M on σ_1, and therefore also $\delta(q_0, \sigma_1) = q$. By the definition of G, there exists a finite computation $\sigma_2 \in$ II' such that $\delta(q_0, \sigma_2) = q$. Let $\sigma' \in \Sigma^\omega$ be the suffix of σ following σ_1, i.e., $\sigma = \sigma_1 . \sigma'$. Denote by r_1 the run of M over $\sigma = \sigma_1 . \sigma'$, and by r_2 the run of M over $\sigma_2 . \sigma'$. Obviously, r_1 and r_2 can differ only by a finite prefix. $\mathcal{M}$ accepts $\sigma_2 \cdot \sigma'$ because $\sigma_2 \in$ II'. Since $Inf_{\mathcal{M}}(r_1) = Inf_{\mathcal{M}}(r_2)$, M must also accept $\sigma_1 \cdot \sigma' = \sigma$. Thus $\sigma \in$ II.

Assume that $\sigma \in$ II. There must exist a prefix $\sigma' \prec \sigma$ such that $\sigma' \in$ II'. Let σ' be the minimal such prefix of σ. Let $q = \delta(q_0, \sigma')$. Obviously $q \in$ G, and q is the first G-state that $\mathcal{M}$ visits on reading σ. It follows that also $q = \delta'(q_0, \sigma')$. By the way M' is constructed, once it reaches a G-state it stays there forever. Consequently, M' accepts σ.

■ Next, consider the case that II is a recurrence property. This means that II = $R_\omega(\Pi')$ for some finitary II'.

We perform a series of modifications on the individual pairs of sets R_i, P_i, $i = 1, \ldots, k$, until all the $P_i' = \emptyset$. These modifications will preserve the property defined by the automaton M.

Without loss of generality, we define the modifications on the first pair R_1, P_1. After obtaining a $P_1' = \emptyset$, we move on to the other pairs.

Assume that all the states in the automaton are reachable. A cycle $\mathcal{C}$ in the automaton is a set of states such that there exists a cyclic path in the automaton that passes only through the states in C, and at least once through each of them. We only consider *accessible* cycles. These are cycles such that the path leading from q_0 to some q in $\mathcal{C}$ and the cyclic path traversing $\mathcal{C}$ are accessible, i.e., never pass through transitions such that $t(q_i, q_j) \equiv$ *false.* A *good* cycle is a cycle such that a run r with $Inf(r) = \mathcal{C}$ is accepting. A *persistent* cycle is a good cycle $\mathcal{C}$ such that $\mathcal{C} \cap R_1 = \emptyset$. Define A_1 to be the set of automaton-states participating in persistent cycles.

Consider the automaton *M'* coinciding with *M* in all but the set of accepting pairs. The list of accepting pairs for *M'* is (R_1', P_1'), *(R;, P_i)*, *i = 2,. . . ,* k, where we define:

$$R_1' = R_1 \cup A_1,$$
$$P_1' = 0.$$

We wish to show that *M* and *M'* accept precisely the same computations.

Consider first a computation σ accepted by *M*. Let J be the infinity set $Inf_{\mathcal{M}}$ (r(o)). Clearly, J satisfies the requirements presented by *(R;, P_i)*, *i > 1,* in both automata. The acceptance for *i* = 1 implies that either $J \cap R_1 \neq \emptyset$ or $J \subseteq P_1$. In the first case obviously J **n** $R_1' \neq \emptyset$. In the second case, if J **n** $R_1 = \emptyset$, then J is a persistent cycle. It follows that $J \subseteq A_1$, and hence $J \cap R_1' \neq \emptyset$.

Consider, next, an infinite computation σ accepted by M'. We will prove that σ is also accepted by *M*. Assume, to the contrary, that σ is rejected by *M*. Let J be as before. Since $\mathcal{M}'$ accepts σ, $J \cap R_1' \neq \emptyset$. The rejection by *M* implies that $J \cap R_1 = \emptyset$. Hence there must be some $q \in A_1$ in J. Let π be a cyclic path from q to itself precisely traversing J. In order for σ to be rejected by $\mathcal{M}$, J must also contain a state $q' \notin R_1$ U P_1. Since $q \in A_1$ there must exist another cycle J', such that *J'* is a persistent cycle. Let π' be the cyclic path from q to itself precisely traversing *J'*. Let σ' be a. finite computation that causes the automaton to move from q back to q along π' .

The state q and computation σ' have the following property:

For every finite computation σ^* such that $\delta(q_0$, a*) = q, there exists a positive integer n (possibly dependent, on σ^*) such that $\sigma^* \cdot (\sigma')^n$ contains a prefix $\tilde{\sigma}$ such that $\tilde{\sigma} \prec \sigma^* \cdot (\sigma')^n$, $|\tilde{\sigma}| \succ |\sigma^*|$, in II'.

To see this, we observe that the computation σ^* · (a'>" has *J'* as infinity set, and is therefore in II. Consequently, $\sigma^* \cdot (\sigma')^\omega$ must have infinitely many prefices in Π', most of which are longer than σ^*. The shortest of these is a prefix of $\sigma^* \cdot (\sigma')^n$ for an appropriate $n > 0$.

Let now σ_0 be a finite computation such that $\delta(q_0, \sigma_0) = q$, and $\hat{\sigma}$ a finite computation leading the automaton from q to q along π. Consider the following infinite computation:

$$\sigma'' = \sigma_0 \hat{\sigma} (\sigma')^{n_1} \hat{\sigma} (\sigma')^{n_2} \ldots,$$

where the n_j's are chosen so that σ'' has infinitely many prefices in II'. That is, for each

$$\sigma_{j-1}'' = \sigma_0 \hat{\sigma} (\sigma')^{n_1} \ldots (\sigma')^{n_{j-1}} \hat{\sigma},$$

we choose an $n_j > 0$ such that $\sigma_{j-1}'' \cdot (\sigma')^{n_j}$ has a prefix in II', which is longer than σ_{j-1}''.

It follows, on one hand, that, since σ'' has infinitely many prefices in II', CT" ∈ II.

On the other hand, the infinity set corresponding to σ'' is $J \cup J'$, which has an empty intersection with R_1 and at least one state $q' \notin P_1$. It follows that M rejects σ'' which contradicts the assumption that M specifies II.

Consequently, there cannot exist a computation σ which is accepted by M' and rejected by M.

It follows that M' is equivalent to M. We can repeat this process for each $i = 2, \ldots, k$ until we obtain an automaton with all $P_i' = \emptyset$, $i = 1, \ldots, k$.

It only remains to show that such an automaton is equivalent to an automaton with a single R and a single $P = \emptyset$. This is essentially the closure property that states that the intersection of recurrence automata, is equivalent to a single recurrence automaton. The construction of this automaton is similar in spirit to the recurrence formula. for the intersection of recurrence formulas. The automaton detects visits to R_2-states such that the most recent previous visit to an $(R_1 \cup R_2)$-state was in fact a visit to an R_1-state (for $k = 2$).

■ The case of a *persistence* property II, that is specifiable by an automaton, is handled by duality. We consider $\overline{\Pi} = \Sigma^\omega - \Pi$, which can be shown to be a recurrence property also specifiable by an automaton.

By the construction for recurrence properties, there exists a recurrence-automaton

$$\mathcal{M}^* = \langle Q, q_0, T, R, \emptyset \rangle$$

specifying $\overline{\Pi}$. Then the following persistence automaton obviously specifies II:

$$\mathcal{M}' = \langle Q, q_0, T, \emptyset, Q - R \rangle.$$

■ The case of *fairness* properties II specifiable by automata, is handled as follows.

Clearly the role of the set of pairs $\{(R_i, P_i) | i = 1, \ldots, k\}$ is to define the subsets $J \subseteq Q$ such that every computation σ with $Inf(r(\sigma)) = J$ is accepted. Let F denote the family of these sets. Obviously $J \in F \leftrightarrow (R_i \cap J \neq \emptyset$ or $J \subseteq P_i)$ for each $i = 1, \ldots, k$.

A characterization property, that can be derived from Wagner [Wag] (see also [Kam]), is the following:

If M specifies a fairness property, then for each accessible accepting set $J \in F$,

either $A \in F$ for every accessible cycle $A \supseteq J$,
or $B \in F$ for every accessible cycle $B \subseteq J$.

An equivalent statement of this fact is that we cannot have a chain of three accessible cycles

$$B \subseteq J \subseteq A,$$

such that $J \in F$, but $B \notin F$ and $A \notin F$.

According to this characterization we can partition the family of accessible' accepting sets into:

$$F = \{A_1, \ldots, A_m, B_1, \ldots, B_n\},$$

where, for each A_i and an arbitrary accessible cycle X, $A_i \subseteq X \rightarrow X \in F$, and for each B_j and an arbitrary accessible cycle X, $X \subseteq B_j \rightarrow X \in F$.

This leads to the construction of the following automaton:

$$M' = \langle Q', q_0', T', R', P' \rangle,$$

where $Q' = Q \times Q^m \times 2 \times n \times 2$.

Each state $q' \in Q'$ has the following structure:

$$q' = \langle q, q_1, \ldots, q_m, f_R, j, f_P \rangle,$$

where $q \in Q$, $q_i \in A_i$, $i = 1, \ldots, m$, $f_R, f_P \in \{0, 1\}$, and $1 \leq j \leq n$.

We assume that the states of M are ordered in some linear order. For each A_i, $i = 1, \ldots, m$, we define $min(A_i)$ to be the state of A_i appearing first in the linear order. For $q \in A_i$, we define $next(q, A_i)$ to be the first state $\hat{q} \in A_i$ appearing after q in the linear order. If $q \in A_i$ is the last A_i-state in the linear order then $next\ (q, A_i) = min\ (A_i)$.

The role of the different components of q' is as follows:

The state q simulates the behavior of the original automaton. Each $q_i \in A_i$ anticipates the next A_i-state we expect to meet. If the run visits all of the A_i infinitely many times, each anticipated q_i will be matched infinitely many times.

The recurrence flag f_R is set to 1 each time one of the anticipated A_i-states is matched.

The index j checks whether the run of M stays completely within one of the sets $B_1, \ldots, B_n$ from a certain point on. It moves cyclically over 1, . . . , n, and at any point checks whether the next automaton-state is in B_j. If the next automaton-state is in B_j, then j retains its value and the next value of f_P will be 1. Otherwise, j is incremented (modulo n), and the next value of f_P will be 0.

$$q_0' = \langle q_0, min(A_1), \ldots, min(A_m), 0, 1, 1 \rangle.$$

T' is defined as follows:

$$t'(\langle q, q_1, \ldots, q_m, f_R, j, f_P\rangle, \langle \tilde{q}, \tilde{q}_1, \ldots, \tilde{q}_m, \tilde{f}_R, \tilde{j}, \tilde{f}_P\rangle) \equiv$$

$$t(q, \tilde{q}) \wedge$$

$$\bigwedge_{i=1}^{m} \{[(\tilde{q} = q_i) \wedge (\tilde{q}_i = next(q_i, A_i))] \vee [(\tilde{q} \neq q_i) \wedge (\tilde{q}_i = q_i)]\} \wedge$$

$$[(\tilde{f}_R = 1) \equiv \bigvee_{i=1}^{m} (\tilde{q} = q_i)] \wedge$$

$$\{[(\tilde{q} \in B_j) \wedge (\tilde{j} = j)] \vee [(\tilde{q} \notin B_j) \wedge (\tilde{j} = [j \bmod n] + 1)]\} \wedge$$

$$[(\tilde{f}_P = 1) \equiv (\tilde{q} \in B_j)].$$

The first clause in this definition states that the first component q follows the same path that would be followed by the original automaton.

The second clause states that either the newly visited automaton-state $\tilde{q}$ matches the anticipated A_i-state, and we modify q_i to the next A;-state in sequence, or there is no match and q_i remains the same.

The third clause states that f_R is set to 1 iff $\tilde{q}$ matches one of the anticipated A_i-states. If different from 1 it must be 0.

The fourth clause states that, if $\tilde{q}$ belongs to B_j, then j is preserved. Otherwise it is incremented in a cyclic manner.

The last clause states that f_P is set to 1 whenever $\tilde{q}$ is in B_j, and to 0 if $\tilde{q} \notin B_j$.

The acceptance sets are defined by:

$$R' = \{\langle q, q_1, \ldots, q_m, 1, j, f_P\rangle \mid \text{for some } q, q_1, \ldots, q_m, j, f_P\},$$
$$P' = \{\langle q, q_1, \ldots, q_m, f_R, j, 1\rangle \mid \text{for some } q, q_1, \ldots, q_m, f_R, j\}.$$

Let σ be a computation and r' the corresponding run of M' over σ. If r' visits R' infinitely many times, this implies that r, the run of M over σ, visits infinitell many times all the states of some A_i. This shows that $Inf(rj \supseteq A_i$, and hence σ is accepted by M as well as by M'.

If r' stays contained in P' from a certain point on, this means that the value of j is never changed beyond that point, and hence r is contained in B_j from that point on. Again, this means that σ is accepted by M as well as by M'.

A similar argument shows that all computations accepted by M are also accepted by M'. ∎

Deciding the Type of a Property

In this section we consider the following problem:

Given a liveness automaton M, decide whether the property specified by this automaton is an a-property, where $a \in$ *{safety, guarantee, recurrence, persistence, fairness).*

The following proposition gives an answer to this general question:

Proposition

It is decidable whether a given liveness (hyper-fairness) automaton specifies an a-property, for $\alpha \in$ *{safety, guarantee, recurrence, persistence, fairness}.*

Again, for the first four types, the answer has been given by Landweber in [Lan]. For the case of fairness, as well as the classes below it in the hierarchy, it is provided by Wagner in [Wag].

In the context of predicate automata, this question was tackled in [AS3], where a decision procedure is given for safety and pure liveness, which is not covered by the previous results.

Since the decision procedures for the cases we consider here are relatively simple, we repeat them below, using our terminology.

First,, some definitions.

A set of automaton-states $A \subseteq Q$ is defined to be *closed* if, for every $q, q' \in Q$,

$$(q \in A \wedge t(q,q') \not\equiv \textit{false}) \rightarrow q' \in A.$$

The closure $\hat{A}$ of a set of states is the smallest closed set containing A.

For a given liveness automaton $\mathcal{M}$, we define $G = \bigcap_{i=1}^{k} (R_i \cup P_i)$.

- Checking for a *safety* property:
  Let $B = Q - G$. The automaton M specifies a safety property iff $\hat{B} \cap G = \emptyset$.
- Checking for a *guarantee* property:
  M specifies a guarantee property iff $\hat{G} \cap B = \emptyset$.

To check for the other levels of the hierarchy, we define the family of accepting sets F:

F = {J | J is an accessible cycle, $J \cap R_i \neq \emptyset$ or $J \subseteq P_i$ for each i = 1,. . . , $k\}$.

The following are direct consequences of the characterizations in [Wag]:

- Checking for a recurrence property:
  $\mathcal{M}$ specifies a recurrence property *iff*, for every $J \in F$ and every accessible cycle $A \supseteq J$, $A \in F$.
- Checking for a *persistence* property:
  $\mathcal{M}$ specifies a persistence property *iff*, for every $J \in F$ and every accessible cycle $B \subseteq J$, $B \in F$.
- Checking for ☞ *fairness* property:
  $\mathcal{M}$ specifies a fairness property *iff* there do not exist three accessible cycles $B \subseteq J \subseteq A$ such that $J \in F$, but $B, A \notin J$.

As a matter of fact, the methods of [Wag] identify the exact location of a liveness property in the hyper-fairness hierarchy, i.e.,·the minimal k such that the property can be specified by a liveness automaton with $|L| = k$.

According to the characterization, this minimal k is the maximal n admitting a chain of accessible cycles of the form

$$B_1 \subset J_1 \subset B_2 \subset J_2 \subset \cdots \subset J_n,$$

where $B_i \notin F$ and $J_i \in F$ for i = 1,. . . , n.

4. Connections Between Temporal Logic and Automata

Temporal logic and predicate automata have been considered as alternatives for specifying properties of programs. A comparison of their expressive power is considered next.

Proposition

A property specifiable by an a-formula is specifiable by an a-automaton, for α ranging over the five types.

This is based on the following construction, studied in [LPZ] and [Z1].

For each finite set of past formulae $p_1, \ldots, p_k$, it is possible to construct a deterministic automaton $\mathcal{M}$ with a set of states Q and designated subsets $F_1, \ldots, F_k$.

The automaton $\mathcal{M}$ has the property that, for each $i = 1, \ldots, k$, each infinite computation $\sigma \in \Sigma^\omega$, and each position $j \geq 0$,

$$\delta(q_0, \sigma[0..j]) \in F_i \quad \leftrightarrow \quad (\sigma, j) \models p_i .$$

Thus, the automaton M identifies, while reading σ up to position j, which p_i's hold at that position.

Using this basic construction, it is straightforward to build an a-automaton corresponding to an a/-formula.

For example, for the fairness formula $\square$ Opl V $\Diamond\square p_2$, let the automaton mentioned above be $\langle Q, q_o, T\rangle$ with the designated sets F_1 and F_2. Then the corresponding fairness automaton is:

$$\langle Q, q_0, T, F_1, F_2\rangle.$$

In the other direction, not every property specifiable by an automaton can be specified in temporal logic. Only a restricted class of automata, called *counter-free* automata (see [MNP] and [W]), can be translated into temporal logic. A (liveness) automaton is defined to be counter-free if there exists no finite computation σ and state q such that $q = \delta(q, \sigma^n)$ for some $n > 1$, but $\delta(q, \sigma) \neq q$. The existence of such q and σ would have enabled the automaton to count occurrences of σ modulo n.

It has been shown in [Z1] that:

An automaton specifies a. property specifiable by temporal logic iff it is counter-free.

This result can be refined to provide a translation from counter-free α-automata to α-formulae.

Proposition

A property specifiable by a. counter-free a/automaton is specifiable by an α-formula,.

The translation is essentially the one studied in [Z1], but shows that the structure required in an cl-automaton corresponds to the structure required in an α-formula.

It is based on the construction of a past-formula φ_q for each $q \in Q - \{q_0\}$ of a given counter-free semi-automaton $\langle Q, q_0, T\rangle$ (i.e., an automaton without accep t ance conditions). The formula φ_q characterizes all the finite computations leading from q_0 to q, i.e., for each infinite computation $\sigma \in \Sigma^\omega$ and position $j \geq 0$,

$$\delta(q_0, \sigma[0..j]) = q \quad \leftrightarrow \quad (\sigma, j) \models \varphi_q.$$

For example, the formula. corresponding to the (counter-free) fairness automaton (Q, q_0, T, R, $P\rangle$ is:

$$\Box\Diamond\Big(\bigvee_{q\in R} \varphi_q\Big) \;\vee\; \Diamond\Box\Big(\bigvee_{q\in P} \varphi_q\Big).$$

The above two-way translation, subject to counter-freedom, provides a standard reduction of results about automata into the corresponding results about temporal logic.

We illustrate this method on the following case of the proposition.

A fairness property Π that is specifiable by temporal logic is specifiable by a fairness formula.

Proof

Let. φ be the formula specifying II. Using the first translation we construct a counter-free automaton $\mathcal{M}_\varphi$ specifying the fairness property II. The part of the proposition dealing with fairness formulae specifiable by automata, tells us how to effectively- construct a fairness automaton $\widetilde{M}$ that specifies the same property. The construction of $\widetilde{\mathcal{M}}$ only refines the structure of $\mathcal{M}_\varphi$, splitting each state of $\mathcal{M}_\varphi$ into many distinct states, respecting the transitions. It follows that, since $\mathcal{M}_\varphi$ is counter-free, so is $\widetilde{\mathcal{M}}$. We can now use the second translation to construct a fairness formula $\varphi_{\widetilde{\mathcal{M}}}$ specifying II. ┛

This method was used in [Z2] to establish the strict hierarchy for temporal formulae, based on [Kam].

5. Proof Principles

One of the main reasons for separating the properties into classes is the expectation that each class will have an appropriate proof principle that can be applied to verify all properties in that class.

To discuss verification of properties over programs, we introduce a minimal model of a program. The minimal model consists of the following elements:

Y – A finite set of program *variables.* This is a set of variables that the program manipulates and controls. It includes both *data variables,* that are explicitly mentioned in the program text, and *control variables,* such as the current location of execution in the program.

Σ – A set of *states.* Each state $s \in \Sigma$ is an assignment of values to variables. . States assign values to a denumerable set of variables that includes all the program variables Y. For a variable y, we denote by $s[y]$ the value assigned by s to y.

θ A state formula (assertion) whose free variables are in Y. This formula characterizes the *initial states* of the program.

ρ A state formula (assertion), whose free variables are a subset of two copies of the program variables, denoted by Y and Y', respectively. This formula, called the *transition formula,* characterizes the relation holding between a state and a possible successor state, obtained by a single execution step of the program. The variables Y and Y' refer to the values assigned to these variables in the state and its successor, respectively.

A *computation* of such a program is an *infinite* sequence of states,

$$\sigma = s_0, s_1, \ldots,$$

such that $s_0 \models$ 6' and, for each $i \geq 0$, $(s_i, s_{i+1}) \models$ p. The meaning of the second requirement is that p is valid over the interpretation that assigns to each $y \in Y$ the value $s_i[y]$, and to each y' $\in$ Y' the value s_{i+1} [y].

To ensure that only infinite computations are considered (in the simplified framework we assume in this paper), we assume that the formula $\forall \overline{y}$: $\exists \overline{y}'$: ρ is valid. This guarantees that every state has a successor.

For a temporal formula φ and a program A, we denote by $A \models \varphi$ the fact that φ is valid over all computations of the program A. We denote by $A \vdash \varphi$ the fact that φ is *provably* valid.

Interface Rules

There are three *interface* axioms/rules from which all the temporal conclusions about the program's behavior can be drawn.

- *Initialit y*

$$A\ t\text{-}6$$

It states that θ always hold at the first state of an A-computation.

- *Invariance Rule*

  Let φ and ψ be two state-formulae whose free variables range over Y. Denote by φ', ψ' the state-formulae obtained by replacing each free variable y $\in$ Y in φ and ψ by its respective primed version y' $\in$ Y'. This substitution can be expressed by
  $$\varphi' = \varphi[Y'/Y] \text{ and } \psi' = \psi[Y'/Y].$$

  The following proof rule establishes the invariance of φ until an occurrence of ψ.
  $$\frac{(\rho \wedge \varphi \wedge \neg\psi) \rightarrow (\varphi' \vee \psi')}{A \vdash (\varphi \Rightarrow \varphi \,\mathfrak{U}\, \psi)}$$

The conclusion of this rule uses the unless operator $\mathfrak{U}$, which is the weaker form of the *until* operator. The conclusion states that, whenever φ occurs, it will continue to hold until the next ocurrence of ψ, if any. If ψ does not occur then φ must continue to hold for the rest of the computation. The rule requires that we establish by state-reasoning (i.e., without temporal reasoning) the premise that, if two states s and s' (interpretations for Y and Y'. respectively) are related by ρ, and the first satisfies φ A $\neg\psi$, then the second state must satisfy φ V ψ. Obviously, under this premise, as long as ψ does not occur, φ is preserved from each state to its successor.

The invariance rule is often used in a simpler form, which is obtained by taking ψ = *false* in the general form. For this special ψ, $\varphi \wedge \neg\psi$, φ' V ψ', and $\varphi \,\mathfrak{U} \psi$ simplify to φ, φ', and $\square$ p7 respectively. The simplified rule is
$$\frac{(\rho \wedge \varphi) \rightarrow \varphi'}{A \vdash (\varphi \Rightarrow \square\varphi)}$$

As an example for the application of the simpler rule, consider the case in which
$$\rho(y, y'): (y' = Y + 1),$$
describing a program whose only action is to increment y by 1. Let
$$\varphi: (y \geq 0).$$

We can easily establish the premise
$$((y' = Y + 1) \wedge (y \geq 0)) \rightarrow (y' \geq 0).$$

By the rule, we then conclude

$$A \vdash ((y \geq$$ [illegible] $$\Rightarrow$$ [illegible] $$\geq$$ [illegible]

- *One-Step Eventuality Rule*

  Let φ, ψ and ψ' be state-formulae as before.

$$\frac{(\rho \wedge \varphi \wedge \neg\psi) \rightarrow \psi'}{A \vdash (\varphi \Rightarrow \Diamond\psi)}$$

This rule requires the premise stating that, if φ holds in a state s, and ψ does not, then ψ holds in each of the A-successors of s. We can then conclude that any occurrence of φ in an A-computation must be eventually followed by an occurrence of ψ.

In a more general framework, in which finite computations are also considered, we have to add a premise guaranteeing *enableness.* An appropriate premise is $\varphi \rightarrow (\exists\overline{y}' : p)$.

As an example for the application of the rule, consider again the incrementing program, and the state-formulae

$$\varphi: (y = 4), \qquad \psi: (y = 5).$$

We establish the premise

$$((Y' = y + 1) \wedge (y = 4)) \rightarrow (y' = 5),$$

and trivially conclude

$$A \vdash ((y = 4) \Rightarrow \Diamond(y = 5)).$$

Well-Founded Eventuality Rule

Obviously, the one-step eventuality rule is very weak and can be used to establish only the simplest type of eventualities, the ones that can be obtained in a single execution step from one state to the next one.

To derive stronger eventualities we combine this basic rule with the powerful well-founded induction rule for eventualities.

- *Well- Founded Eventuality Rule*

  Let $(W, \prec)$ be a well-founded ordering, and $\varphi(\alpha)$, $\alpha \in W$, a state-formula parametrized by a parameter taken from the domain W of the ordering.

$$\frac{A \vdash \varphi(\alpha) \Rightarrow \Diamond[\psi \vee \exists\beta(\beta \prec \alpha) : \varphi(\beta)]}{A \vdash (\exists\alpha : \varphi(\alpha)) \Rightarrow \Diamond\psi}$$

The premise of the rule states that, if $\varphi(\alpha)$ currently holds, then, eventually, either ψ will be established or φ will hold for a smaller parameter $\beta \prec \alpha$.

This premise is typically established by the one-step eventuality rule. Since the decrease of a well-founded parameter cannot go on indefinitely, the rule concludes that eventually ψ must occur.

Note that this more powerful rule does not explicitly refer to any program-specific constructs, such as θ and p. It relies on the third interface rule to help it establish the premise. This explains the name interface rules we have given those rules, since they are the only ones that explicitly refer to program-specific constructs.

We now consider in turn each of the classes of properties, and give for each class an appropriate proof principle.

Safety

A safety formula, has the general form $\Box p$ for some past formula p (i.e., p contains no future temporal operator). How do we verify that such a formula is valid over all computations of a program A?

Consider first the simple case in which p is a state-formula. The suggested proof method in this case is as follows:

Find a state-formula φ such that $\varphi \rightarrow p$, and prove

(a) $\theta \rightarrow \varphi$,

(b) $A \vdash (\varphi \Rightarrow \Box\varphi)$, using the invariance rule.

We may then conclude

$$A \vdash \Box p.$$

Consider next the more general case, that p is a past-formula. Our proposal for dealing with this case is the following:

Without loss of generality we assume that negations appear only within state-subformulae of p. A subformula p' appearing in p is called a *maximal state-subformula* of p if p' is a state-formula and is not contained in a larger state-formula appearing in p.

Let $p_0, p_1, \ldots, p_m$ be the list of all the subformulae of p which either contain a temporal operator , or are maximal state-subformulae of p. The order in which they are listed is such that, if p_j is a subformula of p_i, then $i < j$. We also take $p_0 = p$. We refer to this list as the *closure* of the formula p.

We now define an extension of the program A, denoted by $\hat{A}$, as follows:

Variables – $\hat{Y} = Y \cup \{ b_0, \ldots, b_m\}$. That is, we augment the original program variables by additional *boolean* variables, $b_0, \ldots, b_m$, one corresponding to each subformula in the closure of p.

The intended purpose of these variables is that, in position $j \geq 0$ of a computation σ, $(\sigma, j) \models p_i$ iff $s_j[b_i] = true$.

States – Σ. Each state assigns values also to the variables $b_0, \ldots, b_m$.

Initial Assertion – $\hat{\theta} = \theta \wedge \tilde{\theta}$. The additional conjunct $\tilde{\theta}$ is a conjunction of clauses, θ_i, $i = 0, \ldots, m$, one for each formula, in the closure. The clauses depend on the structure of the formulae, as follows:

If p_i is a maximal state-subformula, then θ_i is $b_i \equiv p_i$.
If p_i is $\neg p_j$, then θ_i is $b_i \equiv (\neg b_j)$.
If p_i is $p_j \vee p_k$, then θ_i is $b_i \equiv (b_j \vee b_k)$.
If p_i is $\ominus p_j$, then θ_i is $b_i \equiv false$.
If p_i is $p_j \mathcal{S} p_k$, then θ_i is $b_i \equiv b_k$.

The intended purpose of these clauses is to guarantee that $b_i \equiv p_i$ at the first state of the computation.

Transition Assertion – $\hat{\rho} = \rho \wedge \tilde{\rho}$. The additional conjunct $\tilde{\rho}$ is a conjunction of clauses, ρ_i, $i = 0, \ldots, m$, defined as follows:

If p_i is a maximal state-subformula, then ρ_i is $b'_i \equiv p'_i$.
If p_i is $\neg p_j$, then ρ_i is $b'_i \equiv (\neg b'_j)$.
If p_i is $p_j \vee p_k$, then ρ_i is $b'_i \equiv (b'_j \vee b'_k)$.
If p_i is $\ominus p_j$, then ρ_i is $b'_i \equiv b_j$.
If p_i is $p_j \mathcal{S} p_k$, then ρ_i is $b'_i \equiv b'_k \vee (b'_j \wedge b_i)$.

The intended purpose of these clauses is to guarantee that, in a transition from a state s to its successor state s', $b_i \equiv p_i$ will be preserved, assuming it already holds at s.

With this augmentation, we can now use the following proof rule:

$$\frac{\hat{A} \vdash \Box b_0}{A \vdash \Box p}$$

Or, if we wish to represent the proof approach in a single rule, it will be of the following form:

To prove $\Box p$, find a state-formula φ, possibly referring to $Y \cup \{ b_0, \ldots, b_m \}$, such that

$$\frac{\begin{array}{c} \varphi \rightarrow b_0 \\ \hat{\theta} \rightarrow \varphi \\ (\hat{\rho} \wedge \varphi) \rightarrow \varphi' \end{array}}{A \vdash \Box p}$$

Example

For example, consider the incrementing program whose only action is to increment y by 1. Assume that its initial assertion $y = 0$, i.e., the initial value of y is 0. We would like to prove for it the safety property $\Box \quad [(y = 10) \rightarrow \diamondminus (y = 5)]$.

We introduce two auxiliary boolean variables b_0, b_1. The first variable b_0, is associated with $(y \neq 10) \vee \diamondminus (y = 5)$, which is the subformula whose invariance we wish to prove. The variable b_1 is associated with $\diamondminus (y = 5)$, which can be represented as $true\,\mathcal{S}\,(y = 5)$. The general construction calls for two more variables, b_2 and b_3, corresponding, respectively, to the maximal state-subformulae $y \neq 10$ and $y = 5$. But in practice, we can skip these variables and refer to the subformulae directly.

$\hat{\theta} = \theta \wedge \tilde{\theta}$ is given by the conjunction:

$$(y = 0) \wedge (b_0 \equiv [(y \neq 10) \vee b_1]) \wedge (b_1 \equiv (y = 5)).$$

$\hat{\rho} = \rho \wedge \tilde{\rho}$ is given by the conjunction:

$$(y' = y + 1) \wedge (b_0' \equiv [(y' \neq 10) \vee b_1']) \wedge (b_1' \equiv [(y' = 5) \vee b_1]).$$

As our assertion, we pick

$$\varphi\colon b_0 \wedge (y \geq 5 \rightarrow b_1).$$

We first show that

$$\hat{\theta} \to \varphi.$$

This is because $y = 0$ implies $y \neq 10$ and $y < 5$.

Next we show that

$$(\hat{\rho} \wedge \varphi) \to \varphi'.$$

By $\hat{\rho}$, b'_0 can be *false* only if $y' = 10$, which is possible only if $y = 9$. But then, due to the clause $(y \geq 5 \to b_1)$ in φ, $b_1 \equiv$ true and therefore, due to the last clause in $\hat{\rho}$, $b'_1 \equiv$ *true,* which leads to $b'_0 \equiv$ *true.*

To show that $(y' \geq 5) \to b'_1$, given that $(y \geq 5) \to b_1$, we should consider the case $y' \geq 5$ while $y < 5$. By $y' = y + 1$, this is possible only if $y' = 5$, which, due to $b'_1 \equiv [(y' = 5) \vee b_1]$, gives $b'_1 \equiv$ *true.* ▟

Guarantee

A guarantee formula has the general form $\Diamond p$ for some past-formula p.

To verify that such formula is valid over a program A we recommend the following approach:

Use the two eventuality rules discussed above to prove

$$A \vdash (\theta \Rightarrow \Diamond p).$$

Then conclude

$$A \vdash o\ p\ .$$

Strictly speaking, this approach covers only the case that p is a. state-formula,. To handle the case of a general past-formula, we augment A as before. and then prove instead

$$\hat{A} \vdash (\theta \Rightarrow \Diamond b_0).$$

Example

For example, we may wish to prove, for the incrementing program, the validity of the guasantee property

$$\Diamond((y = 10)\ \text{A}\ \diamondminus (y = 5)).$$

We rewrite the past-formula using the boolean variables b_0 and b_1 as done above. The variable b_0 corresponds to the principal subformula $(y = 10) \wedge \diamondminus (y = 5)$, while b_1 corresponds to $\diamondminus (y = 5)$. As a matter of fact, the minimal augmentation of the assertions associated with the program deals only with b_1, while we replace the principal subformula by $(y = 10) \wedge b_1$.

The extended initial and transition assertions are given by:

$$\hat{\theta}\colon (y = 0) \wedge (b_1 \equiv (y = 5)),$$
$$\hat{\rho}\colon (y' = y + 1) \wedge (b_1' \equiv [(y' = 5) \vee b_1]).$$

We choose the parametrized assertion:

$$\varphi(n)\colon (n \geq 0) \wedge \quad (y + n = 10) \wedge \quad (y \geq 5 \rightarrow b_1),$$

with n ranging over the well-founded domain of the natural numbers.

We then prove

$$\hat{A} \vdash (\varphi(n) \wedge \neg[(y = 10) \wedge b_1]) \Rightarrow \Diamond \exists m (m < n)\colon \varphi(m)$$

by one-step eventuality.

The conclusion by well-founded eventuality is

$$\hat{A} \vdash \exists n\colon \varphi(n) \Rightarrow \Diamond[(y = 10) \wedge b_1].$$

It only remains to show that $\hat{\theta} \rightarrow \exists n\colon \varphi(n)$, which is obvious by taking $n = 10$, and observing that $y = 0$. ┛

We can show that, for proving guarantee properties, it is sufficient to consider the natural numbers as the well-founded ordering.

Recurrence

A recurrence formula has the general form $\square \Diamond p$ for some past-formula. p.

For the case that, p is a state-formula, we recommend the following methodology:

Find a. state-formula φ and prove

(a) $A \vdash \square\varphi$,
(b) $A \vdash \varphi \Rightarrow \Diamond p$.

Then conclude

$$A \vdash \Box\Diamond p.$$

For example, for the incrementing program, we may wish to prove the recurrence property $\Box\Diamond$ (y *mod* 10 = 0). This property states that, in infinitely many states, y is evenly divisible by 10.

For that simple case it is sufficient to take φ: true, and simply prove, using the well-founded eventuality rule,

$$A \vdash (true \Rightarrow \Diamond(y \; mod\, 10 = 0)).$$

For the case that p is a. general past-formula, we augment the program as before, and prove $\hat{A} \vdash \Box\Diamond\, b_0$ by the same approach.

In contrast with proofs of guarantee properties, recurrence properties require more complex well-founded orderings than just the natural numbers. In fact, already for properties expressible by the formula $\Box p \vee \Diamond q$, we need ordinals higher than ω (see [MP]).

Persistence

A persistence formula, has the general form $\Diamond\Box$ *Ip* for some past-formula. p.

For the case that p is a state-formula, we recommend the following methodology:

Find a state-formula φ, such that $\varphi \rightarrow p$, and prove

(a) $A \vdash \Diamond\varphi$
(b) $A \vdash \varphi \Rightarrow \Box$ *y*.

Then conclude

$$A \ t\text{-} \ \Diamond\Box p.$$

For example, for the incrementing program, we may wish to prove the persistence property $\Diamond\Box(y \neq 5)$.

We pick the state formula φ: (y > 5), and easily show that

$$A \vdash \Diamond(y > 5),$$
$$A \vdash ((y > 5) \Rightarrow \Box \; (y>5)).$$

For the case that p is a general past-formula, we augment the program as before and prove $\hat{A} \vdash \Diamond\Box b_0$, using the same approach.

Fairness

A fairness forniula can always be represented as $\Box\Diamond p \to \Box\Diamond q$ for past-formulae p and q.

For the case that both p and q are state-formulae, we recommend the following methodology :

Find a state-formula $\varphi(\alpha)$ parametrized by a well-founded parameter $\alpha \in W$. Prove

(a) $A \vdash \Box(\exists\alpha{:}\varphi(\alpha))$
(b) $A \vdash \varphi(\alpha) \Rightarrow \varphi(\alpha)\,\mathcal{U}\big(q \vee \exists\beta(\beta \prec \alpha){:}\,\varphi(\beta)\big)$
(c) $A \vdash \big(p \wedge \varphi(\alpha)\big) \Rightarrow \Diamond\big(q \vee \exists\beta(\beta \prec \alpha){:}\,\varphi(\beta)\big).$

We may then conclude

$$A \vdash (\Box\Diamond p \to \Box\Diamond q).$$

The case of p and q being general past-formulae is handled again by augmentation of the program. This time, however, we have to augment it by the boolean variables $b_0, \ldots, b_m$ corresponding to the closure of p, as well as by the boolean variables $c_0, \ldots, c_k$ corresponding to the closure of q. We then use the above method to prove

$$\hat{A} \vdash (\Box\Diamond b_0 \to \Box\Diamond c_0).$$

Hyper-fairness

A hyper-fairness formula. is a conjunction of fairness formulae. Therefore, to prove that it is valid over a program, it is sufficient to prove independently the validity of each of the fairness formulae in the conjunction.

In this section we assumed that the program has no implicit fairness requirements associated with it. Any assumed fairness requirement must be explicitly included in the specification. For example, let the program contain two transitions τ_1 and τ_2, and assume we wish to prove the recurrence formula $\Box\Diamond p$ under the assumptions of weak fairness for each of the transitions. Then we should verify the fairness formula.

$$\Box\Diamond p \vee \Diamond\Box(En(\tau_1) \wedge \neg taken(\tau_1)) \vee \Diamond\Box(En(\tau_2) \wedge \neg taken(\tau_2)).$$

This formula states that either p happens infinitely many times, or from a certain point the execution is weakly unfair towards τ_1, or weakly unfair towards τ_2.

Acknowledgement

We thank Martin Abadi, Tom Henzinger, Muli Safra, Fred Schneider, and Lenore Zuck for many fruitful and illuminating discussions and constructive suggestions. We wish to thank Carol Weintraub for typing the manuscript and Yehuda Barbut for drawing the diagram.

References

[AS1] B. Alpern, F.B. Schneider – Defining Liveness, *Information Processing Letters* 21 (10) (1985).

[AS2] B. Alpern, F.B. Schneider – Verifying Temporal Properties without using Temporal Logic, Technical Report TR85-723 (1985), Cornell University.

[AS3] B. Alpern, F.B. Schneider – Recognizing Safety and Liveness, to appear in *Distributed Computing.*

[Kam] M. Kaminski – A Classification of w-Regular Languages, *Theoretical Computer Science* 36 (1985) 217-229.

[L] L. Lamport – Proving the Correctness of Multiprocess Programs, *IEEE Trans. on Software Engineering* SE-7, 1 (1977).

[Lan] L.H. Landweber – Decision Problems for w-Automata, *Mathematical Systems Theory* 4 (1969) 376-384.

[LPZ] 0. Lichtenstein, A. Pnueli, L. Zuck – The Glory of the Past, Workshop on Logics of Programs, Springer-Verlag LNCS (1985) 196-218.

[MNP] R. McNaughton. S. Papert – Counter Free Automata, MIT Press, Cambridge, MA (1971).

[MP] Z. Manna, A. Pnueli – Specification and Verification of Concurrent Programs by $\forall$-Automata, POPL (1987).

[S] A.P. Sistla – On Characterization of Safety and Liveness Properties in Temporal Logic, Proc. 4th Symposium on Principles of Distributed Computing, ACM (1985) 39-48.

[St] R.S. Streett – Propositional Dynamic Logic with Converse, *Information and Control* 54 (1982) 121-141.

[W] P. Wolper – Temporal Logic can be more Expressive, 22nd Symp. on Foundations of Computer Science (1981) 340-348.

[Wag] K. Wagner – On w-Regular Sets, *Information and Control* 43 (1979) 123-177.

[Z1] L. Zuck – Past, Temporal Logic, Ph.D. Thesis, Weizmann Institute (1986).

[Z2] L. Zuck – Private Communication.