# 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. The classes of the hierarchy are characterized through four views: a language-theoretic view, a topological view, the temporal logic view, and an automata view. In the topological view, the considered hierarchy coincides with the two lower levels of the Borel hierarchy, starting with the closed and open sets. 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 relate this classification to the well known *safety-liveness* classification, and show that in some sense the two are orthogonal to one another.

## 1 Introduction

This paper deals with some methodological aspects of the development of correct *reactive systems*. Reactive systems are systems (and programs) whose main role is to maintain an ongoing interaction with their environment, rather than to produce some final result on termination. Such systems should be specified and analyzed in terms of their behaviors, i.e., the sequences of states or events they generate during their operation. The class of reactive systems includes programs such as operating systems, programs controlling industrial plants, embedded systems, and many others. It is clear that it includes also the classes of concurrent and distributed programs since, independently of the goal and purpose of the complete system, each component of the system has to be studied in terms of the interaction it maintains with the other components.

A reactive program may be viewed as a generator of *computations* which, for simplicity, we may assume to be infinite sequences of states or events. In the case that the program does terminate, we may always extend the finite computation it has generated by an infinite sequence of duplicate states or dummy events to obtain an infinite computation.

In general, we define a *property* as a set of computations. A program $P$ is said to have the property $\Pi$ if all the computations of $P$ belong to $\Pi$. Several languages and formalisms have been proposed for expressing properties of programs, including the language of temporal logic [Pnu77, Lam83] and the formalism of predicate automata [AS89, MP87].

An important approach to the specification and verification of reactive systems is based on specifying a program by listing several properties, representing *requirements* that the program ought to satisfy. This approach enjoys the advantages of *abstraction* and *modularity*. By abstraction we mean that since the specifier lists separate properties and is not required to show how they can be integrated or worry about how they interact with one another, he is not tempted to overspecify or actually *design* the system. Consequently, this approach leads to specifications which are considerably free of implementation bias.

By *modularity* we mean that a property-list based specification is very easy to modify by dropping, adding or modifying a single property. Also, the process of verifying that a proposed implementation satisfies its specification can be done in a modular fashion, by verifying each property separately.

One of the major drawbacks of the property based approach to specification is that, while it discourage

*This research was supported in part by the National Science Foundation under grants CCR-88-12595, CCR-89-11512, and CCR-89-13641; by the Defense Advanced Research Projects Agency under contract N00039-84-C-0211, by the United States Air Force Office of Scientific Research under contracts AFOSR-90-0057 and 88-0281, and by the European Community ESPRIT Basic Research Action project 3096 (SPEC).

†Department of Computer Science, Stanford University, Stanford, CA 94305

‡Department of Applied Mathematics, Weizmann Institute, Rehovot, Israel

overspecification, it may in some cases lead to *underspecification*. Thus, a constant concern in working with such specifications is that of *completeness*: Have we specified enough properties to guarantee that any implementation will be close enough to our intuitive intent?

A classical example of underspecification is a specification for a mutual exclusion algorithm that specifies the obvious requirement that no two processes reside in their critical sections at the same time, but forgets to require that each interested process will eventually get access to its critical section. A trivial but obviously unsatisfactory implementation of this faulty specification is one in which no process ever gets to the critical section.

A partial remedy to the completeness problem can be provided by a deeper study and detailed classification of the different types of properties. This can at least provide the specifier with a check list of properties that he should consider. For each of the properties types he can ask himself the questions: Is there a property of this type that is relevant to the system I am specifying? Have I already specified it?

A useful and important partition of properties into the classes of *safety* and *liveness* properties has been suggested by Lamport in [Lam77]. The two classes have been informally characterized as:

- A *safety* property states that some bad thing *never* happens.
- A *liveness* property states that some good thing *eventually* happens.

An important advantage of this classification is that each class encompasses properties of similar character. Safety properties typically represent requirements that should be continuously maintained by the system. They often express *invariance* properties of the system. For example, if the bad thing represents violation of a mutual exclusion requirement, then the statement that it never happens ensures that mutual exclusion is continuously maintained. Liveness properties, on the other hand, typically represent requirements that need not hold continuously, but whose eventual (or repeated) realization must be guaranteed. For example, if the good thing represents the situation that a process enters its critical section, then the statement that it eventually happens guarantees the absence of a livelock (or individual starvation).

Thus, we can immediately identify the fault with the incomplete specification considered above as missing the appropriate liveness property. With the addition of this property, the specification becomes complete, and admits only satisfactory solutions.

To draw an analogue from 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 to total correctness which also guarantees termination. For reactive systems, which may never terminate, the spectrum of relevant and useful liveness properties is much richer than the single property of termination. For example, it also includes the guarantee that a certain event occurs *infinitely many times*.

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 (or finite prefixes of infinite behaviors) suffices. The only justification for using temporal logic or equivalent formalisms, which are considerably more complex since they define predicates over *infinite* behaviors, is for expressing liveness properties. Thus a major justification for studying the classification of properties is to identify the tradeoff between completeness and expressibility of the specification language and its complexity.

Another reason for wishing to distinguish between the two classes is that their verification calls for different proof approaches. To prove safety properties one uses essentially *computational induction* (using the terminology of [Man74]), based on an invariance principle. According to this principle, to establish a safety or an invariance property, we show that it holds initially and that it is preserved by each individual action of the program. Consequently, based on an induction argument on the position in the computation, it follows that the property always holds. Note that the inductive argument appears only in the justification of the proof principle, but not in its application. That is, the user of the principle only establishes the two facts mentioned above, i.e., initial validity and preservation over the program steps. Thus the induction is *implicit*.

Liveness properties, on the other hand, are proven using *structural induction*, i.e., *explicit* induction on some function of the state, that measures the distance away from the realization of the "good thing". This induction is often represented as an application of a well-founded argument. We refer the reader to [OL82, MP84] for a discussion of the unique proof principles associated with each class.

A more formal definition of the semantic nature of safety properties has been given in [Lam83], and a semantic characterization of liveness properties has been given in [AS85]. These two formal definitions lead to the following pleasing consequences:

- The classes of safety and liveness properties are disjoint (except for the trivial properties of the empty sets and set of all computations).
- Every property can be represented as the intersection of a safety property and a liveness property.
- The classes contain the obvious properties that are intuitively associated with them, i.e., invariance and partial correctness in the safety class, and termination and absence of individual starvation in the liveness class.
- All the properties classified as safety properties can be proven using an invariance principle, while the properties classified as liveness properties can be proven using a well-founded principle.

Alpern and Schneider (see [AS89, AS87]) studied this classification in terms of predicate automata, a topic we will cover in a later subsection. They provided syntactic characterization of the two classes by imposing structural constraints on the automata describing such properties.

Unfortunately, the situation is somewhat less satisfactory when we try to give a syntactic characterization of these two classes in terms of the temporal logic formulae expressing them. Sistla gave in [Sis85] a syntactic characterization of the safety class. He also gave some characterizations of some subclasses of the liveness class, which however do not cover the full class.

In this paper we present an alternative classification of properties, to which we refer as the *Borel classification*. The name is justified by the correspondence we show between this classification and the lower two levels of the classical Borel topological hierarchy.

The Borel classification agrees with the *safety-liveness* (SL) classification on the identification of the class of safety properties. It differs from the SL classification on the other classes. Unlike the SL classification, the Borel classification is a hierarchy and not a partition. This means that some non-safety classes properly contain the class of safety properties. The situation is similar in that respect to the hierarchy of formal languages where, for example, the class of context-free languages properly contains the class of regular languages. We have to designate a language as *strictly* context-free in order to identify it as context-free but not regular.

It is possible to argue that the main advantages attributed above to the safety-liveness classification, namely, the need for more complex specification language and different proof principle for the liveness class, are due to the distinction between the safety and the non-safety classes. The Borel classification also makes such a distinction, but in addition provides a finer classification of the non-safety classes. This classification enables us to distinguish between properties stating that a certain good thing occurs at least once, and properties stating that a certain good thing occurs infinitely many times in the computations.

We examine the Borel classification from four points of view. The first view is linguistic (language theoretic), where we characterize the different classes according to the way they can be constructed from languages of finite sequences. The second view is topological, where we characterize the classes as sets with particular topological properties. This is where we establish the correspondence with the Borel topological hierarchy. Next, we consider properties which are expressible in temporal logic, and give for each class a syntactic characterization of the formulae that express properties belonging to that class. Finally, we consider the expression of properties by predicate automata, and give syntactic characterization of the automata describing properties belonging to each class. We show that these four views of the hierarchy coincide. In the last section dealing with the program part of the proof system, we will indicate some unique proof principles that correspond to particular classes.

A hierarchy, very similar to the one considered here, has been studied extensively in the context of automata over infinite words, which is the fourth 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 [Lan69]. The complete hierarchy has been analyzed in [Wag79], and several years later in [Kam85]. They have also established the connection to the topological characterization. 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 ([Zuc86]).

Parts of this paper have appeared in other places. Some of the main results have been presented in PODC87, but the proceedings of that year contains only an abstract. Other parts have appeared in [MP89a].

## 2 The Linguistic View

In this paper we take an abstract view of the states that a program may assume, the computations that a program may generate, and of the properties that programs may possess. We consider a fixed set of states $\Sigma$, with no assumptions about their internal structure. Computations will correspond to infinite sequences of states. Obviously, a property is a *predicate* on such sequences. It judges some sequences to be acceptable (having the property) and other sequences as unacceptable (not having the property). Thus, each property uniquely defines a characteristic *set* of sequences which are precisely the sequences that have the property.

This leads to a most abstract view of a property as a set of infinite sequences. We denote by $\Sigma^*$ the set of all finite sequences of states in $\Sigma$, and by $\Sigma^+$ the set of all *non-empty* finite sequences of states. Let $\Sigma^\omega$ denote the set of all *infinite* sequences of states, and $\Sigma^\infty = \Sigma^+ \cup \Sigma^\omega$ the set of all non-empty finite and infinite sequences of states.

As suggested above, we view a property as any set of sequences. Another name for such a set is a *language* over the alphabet $\Sigma$. In formal language theory, it is customarily required that the alphabet be finite. However, the extension to infinite alphabets is straightforward. We will therefore consider the terms property and language to be synonymous. Consistently with this terminology, we will refer to sequences as *words*.

Consider, for example $\Sigma$ for a particular program to consist of states assigning integer values to the variable $x$. Let $\Pi$ be the property requiring that

The value of $x$ is monotonically increasing.

Then the property defines a characteristic set, which we also denote by $\Pi$, such that the sequence

$$\langle x : 0\rangle, \langle x : 2\rangle, \langle x : 3\rangle, \ldots$$

belongs to $\Pi$, while the sequence

$$\langle x : 0\rangle, \langle x : 2\rangle, \langle x : 1\rangle, \ldots$$

does not.

We introduce the following special cases of properties. A set $\Phi \subseteq \Sigma^+$ of non-empty finite words is called a *finitary* property. A set $\Pi \subseteq \Sigma^\omega$ of infinite words is called an *infinitary* property. Clearly, our ultimate interest is in the infinitary properties, as all computations are infinite sequences of states. However, the theory of infinitary properties makes extensive use of finitary properties as the building blocks from which infinitary properties are constructed.

For a finite word $\sigma \in \Sigma^+$ and a word $\sigma' \in \Sigma^\infty$, we denote by $\sigma \prec \sigma'$ the fact that $\sigma$ is a *proper finite prefix* of $\sigma'$, i.e., a prefix that differs from $\sigma'$. We denote by $\sigma \preceq \sigma'$ the more general relation $(\sigma \prec \sigma') \vee (\sigma = \sigma' \in \Sigma^+)$ , still requiring $\sigma$ to be finite. The word $\sigma \cdot \sigma'$ is obtained by concatenating $\sigma'$ to the end of $\sigma$. It is defined only if $\sigma$ is finite.

For a property $\Pi \subseteq \Sigma^\infty$, we denote by $Pref(\Pi)$ the set of all finite prefixes of $\Pi$.

We define the complements of a finitary property $\Phi$ and of an infinitary property $\Pi$, denoted respectively by $\overline{\Phi}, \overline{\Pi}$, as

$$\overline{\Phi} = \Sigma^+ - \Phi, \quad \overline{\Pi} = \Sigma^\omega - \Pi.$$

The classification of properties in the linguistic view is based on the following question: How can we construct infinitary properties from finitary ones? The underlying assumption is that finitary properties are easy to understand and handle, but we want to study carefully the construction of infinitary properties.

We propose four operators for the construction of infinitary properties form finite ones. They are denoted by $A$, $E$, $R$, and $P$, respectively. We present below the definition of the properties that are obtained by applying the operators to a given finitary property $\Phi$. We will illustrate these definitions on simple cases, which are described by the notation of regular expressions, extended by the notation $\Phi^\omega$ that denotes the infinite product of the language $\Phi$. Thus, the language $\Phi^\omega$ consists of all the infinite words that can be presented as the infinite concatenation

$$\sigma_0 \cdot \sigma_1 \cdot \sigma_2 \cdots,$$

where each $\sigma_i$ is a finite non-empty word belonging to $\Phi$.

- The property $A(\Phi)$ consists of all the infinite words $\sigma$, such that

  *All* prefixes of $\sigma$ belong to $\Phi$.

  For example, if $\Phi = a^+b^*$, then $A(\Phi) = a^\omega + a^+b^\omega$.

- The property $E(\Phi)$ consists of all the infinite words $\sigma$, such that

  There *exists* a prefix of $\sigma$ that belongs to $\Phi$.

  For example, $E(a^+b^*) = a^+b^* \cdot \Sigma^\omega$. In fact, it is true for every finitary property $\Phi$ that $E(\Phi) = \Phi \cdot \Sigma^\omega$.

- The property $R(\Phi)$ consists of all the infinite words $\sigma$, such that

  *Infinitely many* prefixes of $\sigma$ belong to $\Phi$.

  For example, $R(\Sigma^* b) = (\Sigma^* b)^\omega$. This language contains all the words that have infinitely many occurrences of $b$.

- The property $P(\Phi)$ consists of all the infinite words $\sigma$, such that

  *All but finitely many* prefixes of $\sigma$ belong to $\Phi$.

  For example, $P(\Sigma^* b) = \Sigma^* b^\omega$. This language contains all the words that from a certain point on contain only occurrences of $b$.

The reason for denoting the two last operators by the letters $R$ and $P$ is that prefixes belonging to $\Phi$ occur *recurrently* in $R(\Phi)$, and *persistently* (from a certain point on) in $P(\Phi)$.

For some of the developments below, it is useful to define the finitary versions of the operators $A$ and $E$. Let $\Phi$ be a finitary property. we define

- $A_f(\Phi)$ is the set of all *finite* words $\sigma$, such that

  *All* prefixes of $\sigma$ are in $\Phi$.

- $E_f(\Phi)$ is the set of all *finite* words $\sigma$, such that

  There *exists* a prefix of $\sigma$ that belongs to $\Phi$.

To illustrate the difference between the operators $A, E$ and their finitary versions, consider the following examples

$$\begin{array}{ll} A_f(a^+b^*) = a^+b^* & A(a^+b^*) = a^\omega + a^+b^\omega \\ E_f(a^+b^*) = a^+b^* \cdot \Sigma^* & E(a^+b^*) = a^+b^* \cdot \Sigma^\omega \end{array}$$

The four operators are not completely independent. In fact $A$ and $E$ are dual operators, and so are $R$ and $P$.

The meaning of duality between $A$ and $E$ is that they satisfy the equalities

$$\overline{A(\Phi)} = E(\overline{\Phi}) \quad \text{and} \quad \overline{E(\Phi)} = A(\overline{\Phi}).$$

where complementation is taken with respect to $\Sigma^+$ for $\Phi$, and with respect to $\Sigma^\omega$ for $A(\Phi)$ and $E(\Phi)$.

Similar duality relations hold for the finitary versions of these operators.

$$\overline{A_f(\Phi)} = E_f(\overline{\Phi}) \quad \text{and} \quad \overline{E_f(\Phi)} = A_f(\overline{\Phi}).$$

Let us show, for example, that the equality $\overline{A(\Phi)} = E(\overline{\Phi})$ holds. Clearly $\sigma \in A(\Phi)$ iff all prefixes of $\sigma$ belong to $\Phi$. Consequently $\sigma \notin A(\Phi)$ iff there exists at least one prefix of $\sigma$, call it $\sigma'$, that does not belong to $\Phi$. This means that $\sigma$ has a prefix, namely $\sigma'$, that belongs to $\overline{\Phi}$, which is true iff $\sigma \in E(\overline{\Phi})$.

The duality between $R$ and $P$ is given by the equalities

$$\overline{R(\Phi)} = P(\overline{\Phi}) \quad \text{and} \quad \overline{P(\Phi)} = R(\overline{\Phi}).$$

Based on these four operators we define four basic classes of infinitary properties.
An infinitary property $\Pi$ is defined to be

- A *safety* property if $\Pi = A(\Phi)$ for some finitary $\Phi$. That is, all prefixes of a word $\sigma \in \Pi$ belong to $\Phi$.

- A *guarantee* property if $\Pi = E(\Phi)$ for some finitary $\Phi$. That is, each word $\sigma \in \Pi$ is *guaranteed* to have some prefix belonging to $\Phi$.

- A *recurrence* property if $\Pi = R(\Phi)$ for some finitary $\Phi$. That is, each word $\sigma \in \Pi$ has *recurrently* (infinitely many times) prefixes belonging to $\Phi$.

- A *persistence* property if $\Pi = P(\Phi)$ for some finitary $\Phi$. That is, each word $\sigma \in \Pi$ has *persistently* (continuously from a certain point on) prefixes belonging to $\Phi$.

We apologize to our readers for the frequent changes in the names we give to the classes (compare, for example, with the names appearing in [MP89b]). At least the definitions and characterization of the classes remain the same.

It follows, for example, that the properties $a^\omega + a^+b^\omega$, $a^+b^* \cdot \Sigma^\omega$, $(\Sigma^* b)^\omega$, and $\Sigma^* b^\omega$, are safety, guarantee, recurrence, and persistence properties, respectively.

If we interpret the "good" and "bad" things that are mentioned in Lamport's informal definition as situations or occurrences that can be detected in finite time, then they must correspond to finitary properties. Consequently, we can view the four classes defined above as making different claims about the frequency of occurrences of "good" things. According to this interpretation, safety, guarantee, recurrence, and persistence claim, respectively, that a "good" thing occurs always, at least once, infinitely many times, or continuously from a certain point on.

## Duality of the Classes

A direct consequence of the duality between the operators $A$ and $E$, and between the operators $R$ and $P$, is a corresponding duality between the classes. This can be expressed by

- $\Pi$ is a safety property iff $\overline{\Pi}$ is a guarantee property.
- $\Pi$ is a recurrence property iff $\overline{\Pi}$ is a persistence property.

## Closure of the Classes

Next, we show that each of the four basic classes is closed under the positive boolean operations, namely, union and intersection. We will consider each class in turn.

### Closure of the Guarantee Class

Let $E(\Phi_1)$ and $E(\Phi_2)$ be two guarantee properties. We wish to show that their union is also a guarantee property. This is based on the equality

$$E(\Phi_1) \cup E(\Phi_2) = E(\Phi_1 \cup \Phi_2),$$

which appears even more convincing when we write $\Phi \cdot \Sigma^\omega$ for $E(\Phi)$. In this representation, the equality above assumes the form

$$\Phi_1 \cdot \Sigma^\omega \cup \Phi_2 \cdot \Sigma^\omega = (\Phi_1 \cup \Phi_2) \cdot \Sigma^\omega.$$

Next, consider the case of intersection. Here, we base our argument on the equality

$$\Phi_1 \cdot \Sigma^\omega \cap \Phi_2 \cdot \Sigma^\omega = (\Phi_1 \cdot \Sigma^* \cap \Phi_2 \cdot \Sigma^*) \cdot \Sigma^\omega,$$

which can easily be verified. Expressing this equality in terms of the operators $E$ and $E_f$, we obtain

$$E(\Phi_1) \cap E(\Phi_2) = E(E_f(\Phi_1) \cap E_f(\Phi_2)).$$

This shows that the intersection of two guarantee properties can be expressed the application of the operator $E$ to the finitary property $E_f(\Phi_1) \cap E_f(\Phi_2)$. It follows that the intersection is also a guarantee property.

### Closure of the Safety Class

The closure of the safety class under intersection and union is established by the following two equalities, that can be derived by duality from corresponding equalities for the guarantee class.

$$\begin{aligned} A(\Phi_1) \cap A(\Phi_2) &= A(\Phi_1 \cap \Phi_2) \\ A(\Phi_1) \cup A(\Phi_2) &= A(A_f(\Phi_1) \cup A_f(\Phi_2)). \end{aligned}$$

### Closure of the Recurrence Class

We consider first the simpler case of union. It is not difficult to see that

$$R(\Phi_1) \cup R(\Phi_2) = R(\Phi_1 \cup \Phi_2).$$

This equality states the obvious fact that a word $\sigma$ contains either infinitely many $\Phi_1$-prefixes or infinitely many $\Phi_2$-prefixes iff $\sigma$ contains infinitely many $\Phi_1 \cup \Phi_2$-prefixes.

For the more difficult case of intersection, we introduce the following definition.

Let $\Phi_1$ and $\Phi_2$ be two finitary properties. We define the *minimal extension* of $\Phi_2$ over $\Phi_1$, denoted by $minex(\Phi_1, \Phi_2)$, to be the set of words $\sigma_2 \in \Phi_2$, such that

- There exists a word $\sigma_1 \in \Phi_1$, such that $\sigma_1 \prec \sigma_2$, i.e., $\sigma_2$ is a proper $\Phi_2$-extension of $\sigma_1$, and
- There is no $\sigma_2' \in \Phi_2$ such that $\sigma_1 \prec \sigma_2' \prec \sigma_2$, i.e., $\sigma_2$ is a *minimal* proper $\Phi_2$-extension of $\sigma_1$.

Clearly, $minex(\Phi_1, \Phi_2) \subseteq \Phi_2$, and is therefore a finitary property.

As an example, let $\Phi_1 = (a^3)^+$ and $\Phi_2 = (a^2)^+$. Then $minex(\Phi_1, \Phi_2)$ is equal to $(a^6)^* a^2 + (a^6)^* a^4$. On the other hand, $minex\big((a^2)^+, (a^3)^+\big) = (a^6)^+ + (a^6)^* a^3 = \Phi_1$.

Now we can express the effect of intersecting two recurrence properties as

$$R(\Phi_1) \ \cap \ R(\Phi_2) \ = \ R\big(minex(\Phi_1, \Phi_2)\big).$$

We show inclusion in the two directions. Consider an infinite word $\sigma \in R(\Phi_1) \cap R(\Phi_2)$. Let $\sigma_0^1 \prec \sigma_1^1 \prec \sigma_2^1 \prec \cdots$ be the sequence of $\Phi_1$-prefixes in $\sigma$. For each $i = 0, 1, \ldots$ let $\sigma_i^2$ be the *shortest* $\Phi_2$-prefix of $\sigma$ that properly contains $\sigma_i^1$. Clearly, $\sigma_i^2 \in minex(\Phi_1, \Phi_2)$ and there are infinitely many of them. It follows that $\sigma \in R\big(minex(\Phi_1, \Phi_2)\big)$.

In the other direction, assume that $\sigma \in R\big(minex(\Phi_1, \Phi_2)\big)$. Let $\sigma_0^2 \prec \sigma_1^2 \prec \sigma_2^2 \prec \cdots$ be the sequence of $minex(\Phi_1, \Phi_2)$-prefixes in $\sigma$. Clearly, by the definition of $minex$, each of them belongs also to $\Phi_2$, which shows that $\sigma \in R(\Phi_2)$. For each $i = 0, 1, \ldots$, let $\sigma_i^1$ be the *longest* proper $\Phi_1$-prefix of $\sigma_i^2$. From the definition of minimal extension is follows that, for each $i = 0, 1, \ldots$, $\sigma_i^2$ is the minimal $\Phi_2$-extension of $\sigma_i^1$. Obviously, $\sigma_0^2 \preceq \sigma_1^1 \preceq \sigma_2^1 \preceq \cdots$ but it remains to show that the containment is strict. Assume it is not, and let $\sigma_j^1 = \sigma_{j+1}^1$ for some $j$. Then we have the relations

$$\sigma_j^1 \ = \ \sigma_{j+1}^1 \ \prec \ \sigma_j^2 \ \prec \ \sigma_{j+1}^2.$$

These show that $\sigma^2_{j+1}$ is not the minimal proper $\Phi_2$-extension of $\sigma^1_{j+1}$, contrary to the definition of $\sigma^1_j$. It follows that the sequence $\sigma^1_0 \prec \sigma^1_1 \prec \sigma^1_2 \prec \cdots$ contains infinitely many distinct elements, and therefore $\sigma \in R(\Phi_1)$.

### Closure of the Persistence Class

Here we utilize duality to derive the following equalities which show the closure of the persistence class under intersection and union.

$$\begin{aligned} P(\Phi_1) \;\cap\; P(\Phi_2) &= P(\Phi_1 \cap \Phi_2) \\ P(\Phi_1) \;\cup\; P(\Phi_2) &= P\big(\overline{minex(\overline{\Phi}_1, \overline{\Phi}_2)}\big). \end{aligned}$$

## Characterization of the Classes

Our definition of the classes is *constructive*. This means that we have shown how each of the classes can be constructed by applying the operators $A$, $E$, $R$, and $P$ to finitary properties. In some cases this definition is not easy to apply directly. Consider, for example, the question: How do you show that the property $(a+b)^*b^\omega$ is *not* a safety property? Going back to the constructive definition, we have to show that there cannot exist a finitary property $\Phi$, such that $(a+b)^*b^\omega = A_\omega(\Phi)$. On the face of it, this does not seem to be an easy task.

It is therefore very helpful to derive some additional characterization for some of the properties, which are independent of the constructive definition. We present such characterization for the lower classes of safety and recurrence.

**Claim**

An infinitary property $\Pi$ is a safety property iff

$$\Pi \;=\; A\big(Pref(\Pi)\big).$$

Consider first the *if* direction. Clearly, $Pref(\Pi)$ is a finitary property, and therefore, if $\Pi = A\big(Pref(\Pi)\big)$ then, by the constructive definition $\Pi$ is a safety property.

Next, consider the *only if* direction. It is easy to see that $\Pi \subseteq A\big(Pref(\Pi)\big)$, since for any word $\sigma \in \Pi$, all the prefixes of $\sigma$ belong to $Pref(\Pi)$.

By the assumption that $\Pi$ is a safety property it can be presented as $\Pi = A(\Phi)$ for some finitary $\Phi$. By the definition of the $A$ operator it follows that $Pref(\Pi) \subseteq \Phi$. Applying the operator $A$, which can be shown to be monotonic, to both sides of this inclusion, we obtain $A\big(Pref(\Pi)\big) \subseteq A(\Phi) = \Pi$. This establishes the other direction of the equality.

We may now use the characterization claim to show that $(a^*b)^\omega$ is *not* a safety property. A simple calculation yields $Pref\big((a^*b)^\omega\big) = (a+b)^+$, from which we get

$$\begin{aligned} A\Big(Pref\big((a^*b)^\omega\big)\Big) \;=& \\ A\big((a+b)^+\big) \;=&\; (a+b)^\omega \;\neq\; (a^*b)^\omega. \end{aligned}$$

We refer to the operator $A\big(Pref(\Pi)\big)$, applied to an arbitrary property $\Pi$, as the *safety closure* of $\Pi$. Thus the claim above can be reformulated by saying that an infinitary property $\Pi$ is a safety property iff $\Pi$ equals its safety closure.

By duality we can immediately obtain a characterization of the guarantee class.

**Claim**

An infinitary property $\Pi$ is a guarantee property iff

$$\Pi \;=\; E\big(\overline{Pref(\overline{\Pi})}\big).$$

Note that the complement of $\Pi$ should be taken with respect to $\Sigma^\omega$, while the complement of $Pref(\overline{\Pi})$ should be taken with respect to $\Sigma^+$.

## Inclusion among the Classes

Another interesting relation among the classes is that of inclusion, which arranges the classes of properties in a hierarchy. The two classes of recurrence and persistence properties are higher up in the hierarchy in the sense that they properly contain the classes of safety and guarantee.

### Recurrence contains Safety and Guarantee

To show containment of the safety class in the recurrence class, we have to show that any safety property $\Pi = A(\Phi)$ can be presented as a recurrence property, i.e., as the application of the operator $R$ to some finitary property. This is easily accomplished by the equality

$$A(\Phi) \;=\; R\big(A_f(\Phi)\big).$$

This equality states that all prefixes of $\sigma$ are in $\Phi$ iff $\sigma$ has infinitely many prefixes $\sigma'$, such that all the prefixes of $\sigma'$ are in $\Phi$.

To show that the containment is strict it suffices to consider the property $(a^*b)^\omega$, which consists of all words whose states are either $a$ or $b$, but have an infinite number of $b$'s. It is easily seen that $(a^*b)^\omega$ is a recurrence property, as it can be presented as $R\big((a^*b)^+\big)$. On the other hand, as shown above, this property is not a safety property.

To show that any guarantee property, presentable as $\Pi = E(\Phi)$, is also a recurrence property, we use the equality

$$E(\Phi) \;=\; R\big(E_f(\Phi)\big).$$

It is easy to agree to the equality once we represent the operators $E$ in their equivalent form

$$\Phi \cdot \Sigma^\omega \;=\; (\Phi \cdot \Sigma^*) \cdot \Sigma^\omega.$$

To show that the inclusion is strict, we use again the property $(a^*b)^\omega$ that has already been shown to be a recurrence property. It only remains to show it is not a guarantee property, using the appropriate characterization claim. For simplicity, we assume that $\Sigma = \{a, b\}$.

$$\begin{aligned} E\Big(\overline{Pref\big(\overline{(a^*b)^\omega}\big)}\Big) \;&=\; E\Big(\overline{Pref\big((a+b)^*a^\omega\big)}\Big) \\ =\;\; E\big(\overline{(a+b)^+}\big) \;&=\; E(\phi) = \phi \;\neq\; (a^*b)^\omega. \end{aligned}$$

**Persistence contains Safety and Guarantee**

By duality, we can use the previous results to show that any safety and guarantee properties are presentable as persistence properties. The equalities on which these presentations are based are

$$A(\Phi) \;=\; P\big(A_f(\Phi)\big)$$

$$E(\Phi) \;=\; P\big(E_f(\Phi)\big).$$

To show the strictness of the containment we may use the property complementing the property used before. This is the property $(a+b)^*a^\omega$. It is easy to present it as a persistence property by the expression $P\big((a+b)^*a^+\big)$. On the other hand, using the characterization claims for the safety and guarantee classes, it can easily been shown that this property belongs to neither of these classes.

## The Compound Classes

The four classes we have introduced are considered to be the *basic* classes. As we have seen, each of the basic classes is closed with respect to the *positive* boolean operations of union and intersection, but taking the complement moves us from each class to its dual (safety $\leftrightarrow$ guarantee and recurrence $\leftrightarrow$ persistence).

There are two additional classes, to which we refer as the *compound* classes, that can be obtained by taking unrestricted boolean combinations of the basic classes.

- The Obligation Class.

  This class can be defined by three equivalent statements as the class obtainable by

  - Unrestricted boolean combinations of safety properties, or
  - Unrestricted boolean combinations of guarantee properties, or
  - Positive boolean combinations of safety and guarantee properties.

- The Reactivity Class.

  This class can be defined as the class obtainable by unrestricted boolean combination of either recurrence properties alone, or persistence properties alone. Alternately, all properties of the reactivity class can be obtained by positive boolean combinations of both recurrence and persistence.

The definitions above display an obvious tradeoff between using unrestricted boolean combinations of a single class or using positive boolean combinations of a class and its dual.

A typical obligation property is given by the expression $a^*b^\omega + \Sigma^* \cdot c \cdot \Sigma^\omega$ which represents a union of the safety property $a^*b^\omega$ and the guarantee property $\Sigma^* \cdot c \cdot \Sigma^\omega$.

We will study some of the properties of the obligation class.

## The Obligation Class

The obligation class obviously contains both the safety and guarantee class. By examining the property $a^*b^\omega + \Sigma^* \cdot c \cdot \Sigma^\omega$ we see that this containment is strict since this property is neither a safety property nor a guarantee property.

To justify the name given to this class, consider the property defined by

$$\Pi : \; A(\overline{\Phi}) \cup E(\Psi).$$

Each word $\sigma$ belonging to this property, either has all of its prefixes taken from $\overline{\Phi}$ or has at least one prefix taken from $\Psi$. Consequently, if $\sigma$ has a prefix belonging to $\Phi$ it must also have a prefix belonging to $\Psi$. Thus, this property represents a conditional obligation that the word will contain a $\Psi$-prefix if it contains $\Phi$-prefix. In comparison, the guarantee property $E(\Psi)$ represents an *unconditional* guarantee that the word will contain a $\Psi$-prefix. The more general property $\bigcap_i (A(\overline{\Phi_i}) \cup E(\Psi_i))$ represent the multiple obligation to have a $\Psi_i$-prefix for every $i$ for which the word contains a $\Phi_i$-prefix.

The obligation class is obviously closed under all three boolean operations. Using the third version of the definition, we can use the distributive rule to bring any boolean combination into a conjunctive normal form

$$\bigcap_{i=1}^{n}(\Pi_0^i \cup \cdots \cup \Pi_{k-1}^i \cup \Pi_k^i \cup \cdots \cup \Pi_{m-1}^i),$$

where $\Pi_0^i, \ldots, \Pi_{k-1}^i$ are safety properties, and $\Pi_k^i, \ldots, \Pi_{m-1}^i$ are guarantee properties. Using now the closure of the safety and guarantee classes under union, we can replace the union $\Pi_0^i \cup \cdots \cup \Pi_{k-1}^i$ by a single safety property $\Pi_S^i$, and the union $\Pi_k^i \cup \cdots \cup \Pi_{m-1}^i$ by a single guarantee property $\Pi_T^i$. It follows that any obligation property can be represented as the intersection

$$\bigcap_{i=1}^{n}\big(A(\Phi_i) \cup E(\Psi_i)\big),$$

for some $n > 0$, and finitary properties $\Phi_1, \Psi_1, \ldots, \Phi_n, \Psi_n$. We refer to this presentation as the *conjunctive normal form* of obligation properties.

In a completely symmetric way we can present each obligation property in a *disjunctive normal form*

$$\bigcup_{i=1}^{n}\big(A(\Phi_i) \cap E(\Psi_i)\big).$$

For most of our applications we will mainly use the conjunctive normal form.

Any of these forms introduces an internal strict hierarchy within the class of obligation properties. We define the subclass $Obl_k$, for $k = 1, \ldots$, to consist of all the properties that have a conjunctive normal form representation with $n = k$. It is not difficult to see that $Obl_k \subseteq Obl_{k+1}$. This is due to the fact that we can always add the trivial conjunct $A(\Sigma^+) \cup E(\Sigma^+)$ to a conjunction of $k$ terms and transform it into a conjunction of $k+1$ terms.

Less obvious is the fact that this is a strict hierarchy. To present a canonical example that establishes this fact we introduce the following definitions. For a property $\Pi_1$, that may contain both finite and infinite words, and a property $\Pi_2$, we define the product $\Pi_1 \cdot \Pi_2$ to consist of all the infinite words of $\Pi_1$ and all the words $\sigma_1 \cdot \sigma_2$ for a finite $\sigma_1 \in \Pi_1$ and any $\sigma_2 \in \Pi_2$.

Let $\Sigma = \{a, b, c, d\}$. Define $\Pi = a^\omega + (a+b)^* \cdot c \cdot \Sigma^\omega$. Then the property

$$[(\Pi + a^*)d]^{k-1} \cdot \Pi,$$

for $k > 0$, belongs to $Obl_k$ but to no lower $Obl_{k'}$, $k' < k$.

A similar hierarchy can be defined based on a disjunctive, rather than a conjunctive normal form. Note that in both hierarchies, the safety and guarantee properties belong to the lowest subclass $Obl_1$, to which we refer as the subclass of *simple obligation* properties.

The last property of the obligation class we wish to discuss is its strict containment in both the recurrence and persistence classes. Observe that the definition of obligation properties as a positive boolean combination of safety and recurrence properties can be recast into an inductive definition as follows.

- Every safety property is an obligation property.
- Every guarantee property is an obligation property.
- If $\Pi_1$ and $\Pi_2$ are obligation properties, then so are $\Pi_1 \cup \Pi_2$ and $\Pi_1 \cap \Pi_2$.

Based on this definition, it is easy to prove by induction that every obligation property is a recurrence property. This is because every safety property and every guarantee property are recurrence properties, and the union and intersection of recurrence properties are, again, recurrence properties. To show that containment is strict, we may use again the property $(a^*b)^\omega$, which is a recurrence property but can be shown not to be an obligation property.

An identical argument shows that the obligation class is contained in the persistence class. The property $(a+b)^*a^\omega$, which is a persistence property, but can be shown not to be an obligation property, shows that containment is strict.

It can also be shown that the obligation class is precisely the intersection of the recurrence and persistence classes, i.e., it contains all those properties that are each both a recurrence and a persistence property.

## The Reactivity Class

There is a very close analogy between the way the obligation class is constructed by boolean combinations of safety and guarantee properties, and the way the reactivity class is constructed by boolean combinations of recurrence and persistence properties. We can therefore transliterate all the properties established for the obligation class into corresponding properties of the reactivity class.

Every reactivity property is presentable in a *conjunctive normal form*

$$\bigcap_{i=1}^{n}\big(R(\Phi_i) \cup P(\Psi_i)\big),$$

for some $n > 0$, and finitary properties $\Phi_1, \Psi_1, \ldots, \Phi_n, \Psi_n$.

Similarly, every reactivity property is presentable in a *disjunctive normal form*

$$\bigcup_{i=1}^{n} \big(R(\Phi_i) \cap P(\Psi_i)\big),$$

for some $n > 0$.

These two presentations support two infinite strict hierarchies of reactivity properties.

A reactivity property that is presentable in a conjunctive normal form with $n = 1$ is called a *simple* reactivity property.

The class of reactivity properties is closed under all the three boolean operations.

Consider a simple reactivity property $\Pi = R(\Psi) \cup P(\overline{\Phi})$. Obviously a word $\sigma$ belongs to $\Pi$ if either it has infinitely many $\Psi$-prefixes, or all of its prefixes, from a certain point on, do not belong to $\Phi$. Consequently, if $\sigma$ contains infinitely many $\Phi$-prefixes it must also contain infinitely many $\Psi$-prefixes. Thus, we may view the $\Psi$-prefixes as a *reaction* to having infinitely many $\Phi$-prefixes. A more general property of the form $\bigcap_i (R(\Psi_i) \cup P(\overline{\Phi_i}))$, can be viewed as a multiple reaction promising infinitely many $\Psi_i$-prefixes for every $i$ such that $\sigma$ contains infinitely many $\Phi_i$-prefixes. In Figure 1, we present a diagram that displays the six classes we have discussed and the containment relations holding between them.

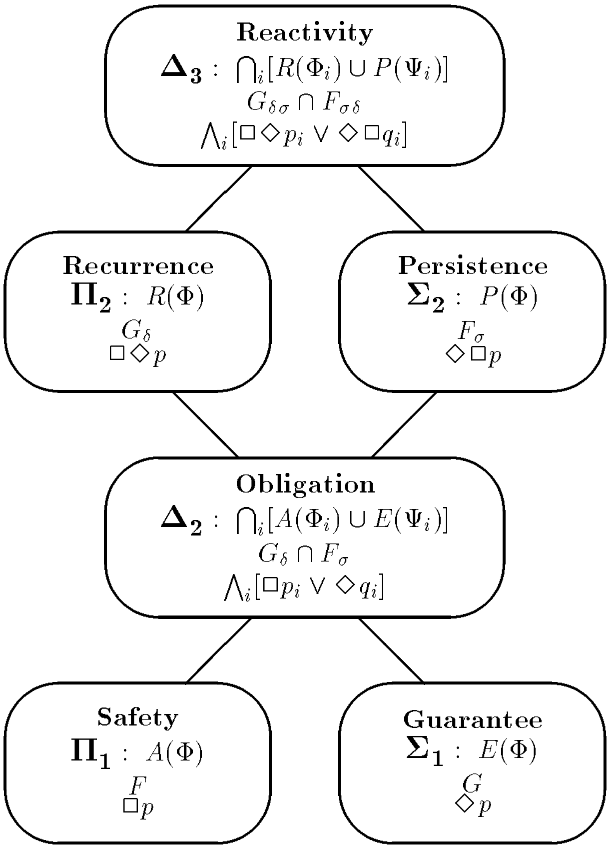


Figure 1: Inclusion Relations between the Classes

## Expression by a First Order Language

The properties $A(\Phi), E(\Phi), R(\Phi)$ and $P(\Phi)$ can be logically characterized as follows. Let $\mathcal{L}$ be the language consisting of individual variables $\sigma, \sigma', \ldots$, unary relations (set symbols) $\Phi, \Psi, \ldots$, and the binary relation $\prec$. Consider properties defined by first-order formulae of the form $\chi(\sigma)$, with a free variable $\sigma$, interpreted in the obvious way.

For each $\mathcal{O} \in \{A, E, R, P\}$, the property $\mathcal{O}(\Phi)$ can be defined by

$$\sigma \in \mathcal{O}(\Phi) \quad \Leftrightarrow \quad \models \chi^{\Phi}_{\mathcal{O}}(\sigma),$$

where

$$\begin{aligned}
\chi^{\Phi}_{A}(\sigma) &: \quad \forall \sigma' \prec \sigma \,.\, \Phi(\sigma') \\
\chi^{\Phi}_{E}(\sigma) &: \quad \exists \sigma' \prec \sigma \,.\, \Phi(\sigma') \\
\chi^{\Phi}_{R}(\sigma) &: \quad \forall \sigma' \prec \sigma \,.\, \exists \sigma''(\sigma' \prec \sigma'' \prec \sigma) \,.\, \Phi(\sigma'') \\
\chi^{\Phi}_{P}(\sigma) &: \quad \exists \sigma' \prec \sigma \,.\, \forall \sigma''(\sigma' \prec \sigma'' \prec \sigma) \,.\, \Phi(\sigma'')
\end{aligned}$$

Thus, it is justified to denote the (class of sequences-sets satisfying) property $A(\Phi)$ by the notation $\mathbf{\Pi_1^{\Phi}}$, the property $E(\Phi)$ by $\mathbf{\Sigma_1^{\Phi}}$, the property $R(\Phi)$ by $\mathbf{\Pi_2^{\Phi}}$, and the property $P(\Phi)$ by $\mathbf{\Sigma_2^{\Phi}}$. We omit the superscript $\Phi$ when referring to the general properties over arbitrary sets $\Phi$. The compound properties of obligation and reactivity can now be denoted by $\mathbf{\Delta_2} = \mathbf{\Pi_2} \cap \mathbf{\Sigma_2}$ and $\mathbf{\Delta_3}$ respectively.

## The Safety-Liveness Classification

As we have already mentioned, there exists another classification that partitions the set of all properties into two *disjoint* classes, the class of *safety* and the class of *liveness* (see [Lam83, AS85]).

The definition of the safety class is the one we have used before, or equivalently, the characterization of an infinitary property as being a safety property if it satisfies

$$\Pi \;=\; A\big(Pref(\Pi)\big).$$

An infinitary property $\Pi$ is defined as being a *liveness* property if every finite word $\sigma \in \Sigma^+$ is a prefix of a word in $\Pi$, i.e., $Pref(\Pi) = \Sigma^+$.

Observe that the definition implies that liveness properties are *upwards closed*. This means that if $\Pi$ is a liveness property, then so is any $\Pi' \supseteq \Pi$.

One of the important aspects of the safety-liveness classification is that it provides an exhaustive partition of the class of all properties. Consider the property $a^*b\Sigma^\omega$, whose temporal logic expression is $a\,\mathcal{U}\,b$. This property is certainly not a safety property as it does not equal its safety closure, which is $a^\omega \cup a^*b\Sigma^\omega$ or $a \,\mathsf{U}\, b$ (using the *unless* operator $\mathsf{U}$). Clearly, the property $a\,\mathcal{U}\,b$ can be represented as the conjunction of the two properties $a \,\mathsf{U}\, b$ and $\Diamond b$. We can view $a \,\mathsf{U}\, b$ as representing the safety component of the property, claiming at any point that we have not lost yet the chance of realizing $a\,\mathcal{U}\,b$.

The property $\Diamond b$, which can be represented by the $\omega$-regular expression $\Sigma^* \cdot b \cdot \Sigma^\omega$, is obviously a liveness formula (as seen by $Pref(\Sigma^* \cdot b \cdot \Sigma^\omega) = \Sigma^+$), and it can be viewed as the most distilled non-safety part of the property $a\,\mathcal{U}\,b$ that does not impose any safety constraints.

Clearly the classes of safety and liveness are disjoint, except for the trivial property $\Sigma^\omega$ (T in temporal logic). Thus, the important fact about the safety-liveness classification is the following

> **Claim**
>
> Every property $\Pi$ can be represented as the intersection
>
> $$\Pi \;=\; \Pi_S \;\cap\; \Pi_L,$$
>
> where $\Pi_S$ is a safety property and $\Pi_L$ is a liveness property.

To prove this claim we take $\Pi_S$ to be the safety closure of $\Pi$, $\Pi_S = A\big(Pref(\Pi)\big)$. For $\Pi_L$ we take the *liveness extension* of $\Pi$, defined by

$$\mathcal{L}(\Pi) \;=\; \Pi \;\cup\; E\big(\overline{Pref(\Pi)}\big).$$

Thus, $\mathcal{L}(\Pi)$ consists of all the words of $\Pi$ plus all the words that have at least one prefix that cannot be extended to a word of $\Pi$. Since for every finitary $\Phi$, $Pref\big(E(\Phi)\big) \subseteq \Phi$, we can compute

$$\begin{aligned} Pref\big(\mathcal{L}(\Pi)\big) \;&=\; Pref(\Pi) \;\cup\; Pref\Big(E\big(\overline{Pref(\Pi)}\big)\Big) \\ &\subseteq\; Pref(\Pi) \;\cup\; \overline{Pref(\Pi)} \;=\; \Sigma^+. \end{aligned}$$

This shows that $\mathcal{L}(\Pi)$ is a liveness property. Next, we show that $\Pi = \Pi_S \cap \Pi_L$. We use the definitions of $\Pi_S$ and $\Pi_L$ and the distribution of intersection over union to get

$$\begin{aligned} \Pi_S \;\cap\; \Pi_L \;\;=\;\; &\Big[A\big(Pref(\Pi)\big) \cap \Pi\Big] \;\cup \\ &\Big[A\big(Pref(\Pi)\big) \cap E\big(\overline{Pref(\Pi)}\big)\Big] \end{aligned}$$

Since every property is contained in its safety closure, it follows that $\Pi \subseteq A\big(Pref(\Pi)\big)$, and hence $A\big(Pref(\Pi)\big) \cap \Pi = \Pi$. The equality $A(\Phi) \cap E(\overline{\Phi}) = \phi$ is true for every finitary property $\Phi$, in particular for $\Phi = Pref(\Pi)$, which leads to the fact that the second intersection is empty. This shows that

$$\Pi_S \;\cap\; \Pi_L \;=\; \Pi.$$

We can identify within the liveness class the same hierarchy we have previously introduced. Let $\kappa$ stand for the name of any of the five non-safety classes, i.e., guarantee, obligation, recurrence, persistence, or reactivity. We define a property to be a *live* $\kappa$-property if it is a liveness property that also belongs to the $\kappa$ class.

An interesting observation is that if $\Pi$ is a property of class $\kappa$, then its liveness extension $\mathcal{L}(\Pi)$ is a live $\kappa$-property. This is because $\mathcal{L}(\Pi)$ is formed by the union of $\Pi$ with the guarantee property $E\big(\overline{Pref(\Pi)}\big)$, and all the non-safety classes are closed under unions with guarantee properties. It follows that

> any property $\Pi$ of the non-safety class $\kappa$ is representable as the intersection
>
> $$\Pi \;=\; \Pi_S \;\cap\; \Pi_L,$$
>
> where $\Pi_S$ is a safety property and $\Pi_L$ is a live $\kappa$-property.

This observation shows that in some sense the Borel and the safety-liveness classifications are orthogonal to one another.

A special case of liveness properties is the class of *uniform liveness* property. A property $\Pi$ is defined to be a *uniform liveness* property if there exists a *single* infinite word $\sigma' \in \Sigma^\omega$, such that $\Sigma^+ \cdot \sigma' \subseteq \Pi$. That is, for every arbitrary $\sigma \in \Sigma^+$, $\sigma \cdot \sigma' \in \Pi$. Note, in comparison, that liveness only requires that for every $\sigma \in \Sigma^+$, there exists *some* $\sigma' \in \Sigma^\omega$, such that $\sigma \cdot \sigma' \in \Pi$, while uniform liveness insists that the same $\sigma'$ extends any finite word to a $\Pi$-word.

As an example of a property that is a liveness property but not a uniform liveness, consider an alphabet $\Sigma = \{a, b\}$. The property $a \cdot \Sigma^* \cdot aa \cdot \Sigma^\omega + b \cdot \Sigma^* \cdot bb \cdot \Sigma^\omega$ requires that the state that appears first in the word appears sometimes later, twice in succession. Let $\sigma$ be any word. If $\sigma$ begins in an $a$-state, then the proper extension is $\sigma' = aa \cdot \Sigma^\omega$. If $\sigma$ begins in a $b$-state, then the proper extension is $\sigma' = bb \cdot \Sigma^\omega$. Clearly, there does not exist a uniform extension $\sigma'$ that applies to all words $\sigma \in \Sigma^+$.

# 3 The Topological View

In this section we characterize the different classes in the hierarchy by their topological properties. We will show that the classes of properties in our hierarchy correspond precisely to the lower two (and a half) levels of the Borel topological hierarchy.

First, let us define the appropriate topological notions. For a word $\sigma$, we denote by $\sigma[i], i = 0, \ldots,$ the $i$'th element (state) of $\sigma$, counting from 0. For example, if $\sigma = 0, 1, 2, \ldots,$ then $\sigma[i] = i$. Also, for any $n \geq 0$, if $\sigma = a^{n+1}b^\omega$, then $\sigma[n] = a$ and $\sigma[n+1] = b$.

We define the *distance* between two infinite words $\sigma$ and $\sigma$ to be 0 if they are identical, and

$$\mu(\sigma, \sigma') = 2^{-j}$$

otherwise, where $j \geq 0$ is the minimal index on which they differ, i.e., such that $\sigma[j] \neq \sigma'[j]$. Equivalently, $j$ is the length of the longest prefix on which they agree. There is nothing magic about this particular function of $j$. Any other function of $j$ that tends to zero as $j$ tends to infinity will do equally well.

For example, for every $n > 0$, $\mu(a^n b^\omega, a^{2n} b^\omega) = 2^{-n}$, since the longest prefix on which these two words agree is $a^n$ of length $n$.

It is not difficult to see that the distance function $\mu$ has all the properties required from a *metric*, and that with this metric, the set $\Sigma^\omega$ becomes a complete metric space.

Following the standard definitions, we say that an infinite sequence of words (each of which is an infinite sequence of states)

$$\sigma_0, \sigma_1, \ldots,$$

*converges* to the *limit* $\sigma$, if the distance $\mu(\sigma, \sigma_k)$ tends to 0 as $k$ goes to infinity. In other words, the length of the maximal prefix common to $\sigma$ and to $\sigma_k$ grows to infinity together with $k$.

For a given integer $L > 0$, we say that the words $\sigma$ and $\sigma'$ *share a prefix longer than* $L$, if there exists a finite word $\hat{\sigma}$, which is a prefix of both $\sigma$ and $\sigma'$, and whose length exceeds $L$. With this notion we can reformulate the definition of the sequence $\sigma_0, \sigma_1, \ldots$ converging to $\sigma$ if for every $L > 0$ there exists an index $k$, such that $\sigma$ and $\sigma_k$ share a prefix longer than $L$. The advantage of this version of the definition is that it does not depend explicitly on the particular distance function $\mu$.

Consider, for example, the sequence of words $b^\omega, ab^\omega, aab^\omega, aaab^\omega, \ldots$. It is not difficult to see that this sequence converges to the limit $a^\omega$. This is because the sequence of longest prefixes common to $a^\omega$ and to $\sigma_k = a^k b^\omega$, which is $a^k$, gets increasingly longer with $k$.

Given a set $U \in \Sigma^\omega$, we define the word $\sigma \in \Sigma^\omega$ to be a *limit point* of the set $U$, if there exists an infinite sequence of words $\sigma_0, \sigma_1, \sigma_2, \ldots,$ all of which belong to $U$, that converges to $\sigma$. Clearly, any $\sigma \in U$ is a limit point of $U$, since the sequence $\sigma, \sigma, \sigma, \ldots$ converges to $\sigma$.

We define the (topological) *closure* of $U$, denoted by $cl(U)$, to be the set of all limit points of $U$. Obviously, $U \subseteq cl(U)$. For example, the closure of the set $a^+b^\omega$ is given by $cl(a^+b^\omega) = a^+b^\omega + a^\omega$, which consists of the original set plus the limit word $a^\omega$.

We define a set $U$ to be *closed* if it contains all its limit points, i.e., $cl(U) \subseteq U$, which immediately yields $cl(U) = U$. Thus, $a^+b^\omega + a^\omega$ is a closed set, while $a^+b^\omega$ is not.

We proceed to consider each of the classes of properties we have introduced above, and give it a topological characterization. We will start with the four basic classes.

### The Safety Properties are the *Closed* sets

To show the correspondent between the safety properties and closed sets, we prove the following general identity that hold for any set $\Pi \subseteq \Sigma^\omega$

$$cl(\Pi) \quad = \quad A(Pref(\Pi)).$$

Let us denote $\Phi = Pref(\Pi)$. The set $\Phi$ is of course *prefix closed*, which means that if $\sigma \in \Phi$ and $\sigma' \prec \sigma$, then also $\sigma' \in \Phi$.

Let $\sigma \in cl(\Pi)$. This means that there exists an infinite sequence $\sigma_0, \sigma_1, \ldots$ of words in $\Pi$, whose limit is $\sigma$. Let $\hat{\sigma} \prec \sigma$ be an arbitrary prefix of $\sigma$. By the definition of a sequence converging to a limit, there must exists some $k \geq 0$, such that $\sigma$ and $\sigma_k$ share a common prefix $\sigma'$ which is *longer* than $\hat{\sigma}$. It follows that $\hat{\sigma} \prec \sigma' \prec \sigma_k$, and therefore $\hat{\sigma} \in Pref(\Pi)$. The case that $\sigma_k = \sigma$ is similar. Thus, any prefix of $\sigma$ is in $Pref(\Pi)$, and therefore $\sigma \in A(Pref(\Pi))$.

In the other direction, let $\sigma \in A(Pref(\Pi))$. For each $k > 0$, let $\hat{\sigma}_k$ denote the prefix of $\sigma$ of length $k$. By the assumption, each $\hat{\sigma}_k$ belongs to $Pref(\Pi)$, and therefore there exists a $\sigma_k \in \Pi$, such that $\hat{\sigma}_k \prec \sigma_k$. We claim that the sequence $\sigma_1, \sigma_2, \ldots$ converges to $\sigma$. This is because for any length $L$ it is sufficient to take $k > L$ to obtain a member of the sequence, namely

$\sigma_k$, which shares with $\sigma$ at least the prefix $\hat{\sigma}_k$, whose length exceeds $L$. Thus, $\sigma$ is a limit point of $\Pi$. ∎

Having established the equivalence of the operators $cl(\Pi)$ and $A(Pref(\Pi))$, it easy to conclude that

> $\Pi$ is a safety property iff $\Pi = A(Pref(\Pi))$ iff $\Pi = cl(\Pi)$ iff $\Pi$ is a closed set.

The closure of the class of closed sets with respect to finite unions and intersections is a known topological fact, and corresponds to the similar closure properties we have established in the linguistic view. In the inclusion diagram we represent the topological characterization of this class by the letter $F$, which usually designate the class of *closed* sets.

## The Guarantee Properties are the *Open* sets

Following the standard topological definitions, a set $\Pi$ is defined to be *open* in our topology if for every $\sigma$ belonging to $\Pi$, there exists an $L > 0$, such that any other word $\sigma'$, sharing with $\sigma$ a prefix longer than $L$, is also in $\Pi$. Thus, a complete environment of all the words that are close enough to $\sigma$ is fully contained in $\Pi$. It is not difficult to see that $\Pi$ is an open set iff $\overline{\Pi}$ (the complement with respect to $\Sigma^\omega$) is closed. Thus the families of closed an open sets are *dual*.

We can use this duality, and the previously established equivalence between safety properties and the family of closed sets to establish the equivalence between guarantee properties and the family of open sets as follows:

> $\Pi$ is a guarantee property iff $\overline{\Pi}$) is a safety property iff $\overline{\Pi}$ is a closed set iff $\Pi$ is an open set.

However, it may be instructive to present an independent proof of the latter equivalence.

Let $\Pi = E(\Phi)$ be a guarantee property. Let $\sigma \in \Pi$ be any word in $\Pi$. By definition, $\sigma$ has a finite prefix $\hat{\sigma} \prec \sigma$ which belongs to $\Phi$. Assume its length to be $L > 0$. Then we claim that all infinite words which share with $\sigma$ a prefix longer than $L$ are also in $\Pi$. Let $\sigma'$ be such an infinite word. Since $\sigma$ and $\sigma'$ share a prefix longer than $L$, $\sigma'$ must also have $\hat{\sigma}$ as a prefix. Consequently, $\sigma' \in \Pi$.

In the other direction, Let $\Pi$ be an open set. Take any infinite word $\sigma \in \Pi$. Since $\Pi$ is open, there must be some integer $L > 0$, such that all infinite words that share with $\sigma$ a prefix longer than $L$ are in $\Pi$. Let $\hat{\sigma}$ be the prefix of length $L + 1$ of $\sigma$. We define $\Phi$ to be the set of all such prefixes. It is not difficult to see that $E(\Phi) = \Pi$. ∎

In the inclusion diagram, we denote the characterization of the guarantee properties as the family of open sets by the designation $G$ which stands for the open sets.

## The Recurrence Properties are the $G_\delta$ sets

A set is defined to belong to the $G_\delta$ family if it can be obtained as a countable intersection of open sets. Note, that if we take only a *finite* intersection of open sets, we still obtain an open set.

Consider for example the sequence of open sets

$$G_1 : (a^*b)\cdot\Sigma^\omega \ , \ G_2 : (a^*b)^2\cdot\Sigma^\omega \ , \ G_3 : (A^*b)\cdot\Sigma^\omega \ , \ \ldots,$$

where we assume $\Sigma = \{a, b\}$. Clearly $G_k$ is the set of all words that have at least $k$ occurrences of the letter $b$. It is not difficult to see that the intersection of the sequence of these sets yields the set $H : (a^*b)^\omega$, that consists of all the words having each an infinite number of $b$'s. It can easily be shown that the set $H$ is neither open nor closed, but is, by definition a $G_\delta$ set.

Let us show now that a set $\Pi$ is a recurrence property iff it is a $G_\delta$ set.

Assume, first, that $\Pi$ is a recurrence property, i.e., $\Pi = R(\Phi)$ for some finitary $\Phi$, and let $\sigma \in \Pi$. For every $k > 0$ define the set $\Phi_k$ to be the set of words $\sigma$ such that $\sigma \in \Phi$, and $\sigma$ contains $k - 1$ distinct proper prefixes that belong also to $\Phi$. Some of the sets $\Phi_k$ may be empty, but they cannot all be empty, otherwise $\Pi$ would have been empty. For each $k > 0$ define now $G_k = \Phi_k \cdot \Sigma^\omega = E(\Phi_k)$. It is not difficult to see that the $G_k$'s are open, and that

$$\Pi = \bigcap_k G_k.$$

In the other direction, let $\Pi$ be a $G_\delta$ set, i.e., $\Pi = \bigcap_k G_k$ for some open sets $G_0, G_1, \ldots$. By the characterization of open sets, each $\Box_k$ can be represented as $G_k = \Phi_k \cdot \Sigma^\omega$ for some finitary $\Phi_k$. Consider an arbitrary $\sigma \in \Pi$. For every $k \geq 0$, $\sigma$ must have a prefix $\hat{\sigma}_k$ belonging to $\Phi_k$. Without loss of generality we may assume that $\hat{\sigma}_k$ is the shortest preix of $\sigma$ that belongs to $\Phi_k$. There are two cases to be considered. In the first case, the lengths of the prefixes $\hat{\sigma}_k$ are bounded. In that case there exists a single prefix $\hat{\sigma}$ such that $\hat{\sigma}_k \preceq \hat{\sigma}$. Defining a finitary set $\Phi = \bigcap_k(\Phi_k \cdot \Sigma^\omega)$, we can easily see that $\hat{\sigma}$ belongs to $\Phi$. Thus, in the bounded case, $\sigma$ belongs to $\Phi \cdot \Sigma^\omega = E(\Phi)$. Next consider the case that the lengths of the prefixes $\hat{\sigma}_k$ are bounded. Let us define a new set of prefixes $\sigma'_0, \sigma'_1, \ldots$, where $\sigma'_k$ is the shortest prefix of $\sigma$ such that $\hat{\sigma}_0 \preceq \sigma'_k, \ldots, \hat{\sigma}_k \preceq \sigma'_k$. It is not difficult to see that the lengths of the $\sigma'_k$ are also unbounded. Define the sets $\Psi_k = \bigcap_{i=0}^{i=k}(\Phi \cdot \Sigma^*)$. Let $\Gamma_k$ be the *prefix free* subset of $\Psi_k$, i.e., the set of words $\hat{\sigma}$ belonging to $\Psi_k$, such that no proper prefix of $\hat{\sigma}$ belongs to $\Psi_k$. Clearly, each $\sigma'_k$ belongs to $\Gamma_k$. If we denote the union of the $\Gamma_k$'s by $\Gamma = \bigcup_k \Gamma_k$ then, in the unbounded case, $\sigma \in R(\Gamma)$ since $\sigma$ has unboundedly many prefixes belonging to $\Gamma$. We may summarize the two cases by

$$\Pi \subseteq E(\Phi) \cup R(\Gamma),$$

for the finitary sets $\Phi$ and $\Gamma$ defined above.

It is not difficult to see that $E(\Phi) \subseteq \Pi$. For the second component, consider some word $\sigma \in R(\Gamma)$. This word has infinitely many prefixes, each belonging to some $\Gamma_k$. Since the sets $\Gamma_k$ are prefix free, the infinitely many prefixes must belong to infinitely many different $\Gamma_k$'s. For any $i \geq 0$, there must be a prefix $\hat{\sigma} \prec \sigma$ belonging to $\Gamma_k$ for $k \geq i$. By the definitions of $\Gamma_k$ and $\Psi_k$, it follows that $\hat{\sigma} \in \Phi_i \cdot \Sigma^*$. Thus $\sigma \in \Phi_i \cdot \Sigma^\omega$ for every $i$. Consequently, $R(\Gamma) \subseteq \Pi$. We may conclude

$$\Pi = E(\Phi) \cup R(\Gamma).$$

Since the union of a guarantee property with a recurrence property is a recurrence property, we conclude that every $G_\delta$ set is a recurrence property. ∎

### The Persistence Properties are the $F_\sigma$ sets

A set is defined to belong to the $F_\sigma$ family if it can be obtained as the countable union of closed sets. Obviously, a set $U$ is $F_\sigma$ iff its complement $\overline{U}$ is $G_\delta$. Since the complement of a persistence property is a recurrence property, the equivalence between persistence properties and $F_\sigma$ sets follows by duality from the corresponding results for recurrence properties.

### Topological Characterization of the Safety-Liveness Classification

Following [AS85], we provide a topological characterization of the safety-liveness classification. We have already shown that the safety class corresponds to the family of closed sets.

As claimed in [AS85], the class of liveness properties corresponds to the family of *dense* sets. A set $U$ is defined to be dense in a space, such as $\Sigma^\omega$, if for any word $\sigma \in \Sigma^\omega$ and any integer $L > 0$, there exists a word $\sigma'$ belonging to $U$, sharing with $\sigma$ a prefix longer than $L$. Thus, members of $U$ exists arbitrary close to any word $\sigma \in \Sigma^\omega$.

To show that every liveness property is dense, consider a liveness property $\Pi$, and let $\sigma$ be an arbitrary word, and $L > 0$ an arbitrary integer. Let $\hat{\sigma}$ be the prefix of $\sigma$ of length $L+1$. By the definition of a liveness set there exists some word $\sigma'$ belonging to $\Pi$, having $\hat{\sigma}$ as a prefix. Consequently, $\sigma$ and $\sigma'$ share a prefix (at least $\hat{\sigma}$), longer than $L$.

## 4 The Temporal Logic View

After studying properties and their classification in an abstract language - theoretical and topological settings, we consider a subset of all these properties, the properties expressible by temporal logic. We will show that the hierarchy introduced in the abstract setting, still exists within the set of expressible properties, and each class has an additional characterization by formulae of a special form that can express all the expressible properties belonging to that class.

As we will see, each class enjoys certain closure properties and is associated with an appropriate proof principle for verifying that a given program satisfies a property in the class.

Our main interest is in the question of what types of properties are expressible in temporal logic (or equivalent formalisms), and whether temporal logic is powerful enough to express all the interesting properties of reactive systems.

We will consider each of the basic concepts introduced under the abstract setting and show how to restrict them to the framework of expressible properties.

First, we present a short introduction to the language of temporal logic.

### The Language of Temporal Logic

We assume an underlying assertional language, which is used to describe properties of states. in the current discussion, we will consider two types of state languages. For the case that $\Sigma$ is finite, we may use the states themselves as basic propositions.

For example, assume that $\Sigma = \{a, b, c\}$. Then, we may take $a$, $b$, and $c$ as basic propositions, that can be combined by the usual boolean combinations. Thus, the assertion $a$ is true on the state $a$ and is false on the states $b$ and $c$. On the other hand, the assertion $\neg a$, which in our case is also equivalent to $b \vee c$ is true on both the states $b$ and $c$, and false on the state $a$.

The other type of state language we will consider assumes that the states have structure, and represent intermediate situations in the computation of a concrete programs. In this case, we assume that the state contains the current values of all the program variables, as well as an appropriate representation of the control of the program, which tells us at what location the program is currently executing. Thus, we will allow basic state-predicates of the form $at\_\ell$, that tell us that the program is currently executing at location $\ell$, as well as explicit references to the current values of the program variables. We will use this version of the state language to illustrate the utility of temporal logic for expressing properties of concrete programs.

We refer to a formula in the assertional language as a *state formula*, or simply as an *assertion*.

A *temporal formula* is constructed out of state formulae to which we apply the boolean operators $\neg$ and $\vee$ (the other boolean operators can be defined from these), and the following basic temporal operators:

$\bigcirc$ – Next $\quad \circleddash$ – Previous
$\mathcal{U}$ – Until $\quad \mathcal{S}$ – Since

A *model* for a temporal formula $p$ is an infinite sequence of states (i.e., a word)

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

where each state $s_j$ provides an interpretation for the state subformulae mentioned in $p$.

Given a model $\sigma$, as above, we present an inductive definition for the notion of a temporal formula $p$ holding at a position $j \geq 0$ in $\sigma$, denoted by $(\sigma, j) \models p$. For a state formula $p$,

$$(\sigma, j) \models p \quad \Longleftrightarrow \quad s_j \models p.$$

That is, we evaluate $p$ locally, using the interpretation given by $s_j$.

$$
\begin{array}{lll}
(\sigma, j) \models \neg p & \Longleftrightarrow & (\sigma, j) \not\models p \\
(\sigma, j) \models p \vee q & \Longleftrightarrow & (\sigma, j) \models p \text{ or } (\sigma, j) \models q \\
(\sigma, j) \models \bigcirc p & \Longleftrightarrow & (\sigma, j+1) \models p \\
(\sigma, j) \models p \mathcal{U} q & \Longleftrightarrow & \text{for some } k \geq j, (\sigma, k) \models q, \\
\multicolumn{3}{l}{\quad \text{and for every } i \text{ such that } j \leq i < k, (\sigma, i) \models p} \\
(\sigma, j) \models \circleddash p & \Longleftrightarrow & j > 0 \text{ and } (\sigma, j-1) \models p \\
(\sigma, j) \models p \mathcal{S} q & \Longleftrightarrow & \text{for some } k \leq j, (\sigma, k) \models q, \\
\multicolumn{3}{l}{\quad \text{and for every } i \text{ such that } j \geq i > k, (\sigma, i) \models p}
\end{array}
$$

Additional temporal operators can be defined as follows:

$$
\begin{array}{ll}
\Diamond p = \mathrm{T} \mathcal{U} p & \text{– Eventually} \\
\Box p = \neg \Diamond \neg p & \text{– Henceforth} \\
p \mathsf{U} q = \Box p \vee (p \mathcal{U} q) & \text{– Unless} \\
\diamondminus p = \mathrm{T} \mathcal{S} p & \text{– Sometimes in the past} \\
\boxminus p = \neg \diamondminus \neg p & \text{– Always in the past} \\
p \mathsf{S} q = \boxminus p \vee (p \mathcal{S} q) & \text{– Weak Since} \\
\widetilde{\circleddash} p = \neg \circleddash \neg p & \text{– Weak Previous}
\end{array}
$$

Another useful derived operator is the *entailment* operator, defined by:

$$p \Rightarrow q \quad \Longleftrightarrow \quad \Box(p \rightarrow q).$$

A formula that contains no future operators is called a *past* formula. A formula that contains no past operators is called a *future* formula. Note that a state formula is both a past and a future formula. We refer to a past formula (future formula) that is not also a state formula, as a *strict-past* (*strict-future*, respectively) formula. For a state formula $p$ and a state $s$ such that $p$ holds on $s$, we say that $s$ is a *$p$-state*.

If $(\sigma, 0) \models p$, we say that $p$ *holds* on $\sigma$, and denote it by $\sigma \models p$. A formula $p$ is called *satisfiable* if it holds on *some* model. A formula is called *valid* if it holds on *all* models.

Two formulae $p$ and $q$ are defined to be *equivalent*, denoted by

$$p \sim q,$$

if the formula $p \equiv q$ is valid, i.e., $\sigma \models p$ iff $\sigma \models q$, for all $\sigma$.

Following are some simple examples of temporal formulae and their intuitive meaning as a requirment on the sequences (words) that satisfy them.

- $p \rightarrow \Diamond q$
  If initially $p$ then eventually $q$.

- $\Box(p \rightarrow \Diamond q)$
  Every $p$-position coincides with or is followed by $q$-position.

- $\Box \Diamond q$
  The sequence $\sigma$ contains infinitely many $q$-positions.

- $\Diamond \Box q$
  Eventually persistently $q$, or equivalently:
  The sequence $\sigma$ contains only finitely many $\neg q$-positions.

- $(\neg q) \mathsf{U} p$
  If there exist any $q$-positions, then the first $q$-position must coincide or be preceded by a $p$-position.

- $\Box(q \rightarrow \diamondminus p)$
  Every $q$-position coincides with or is preceded by a $p$-position.

- $\neg \circleddash \mathrm{T}$
  There is no previous position that satisfies T. Since all positions which are in the model must satisfy T, this is equivalent to:

  There is no previous position.

Note that this formula uniquely characterize the initial position of every model. We refer to this formula as *first*.

## The Temporal Hierarchy of Properties

First, consider the notion of an infinitary property, to which, for simplicity, we will refer simply as a property. In the syntactical framework every property $\Pi \subseteq \Sigma^\omega$ is associated with a temporal formula $\varphi$. The property itself, i.e., the set of sequences belonging to $\Pi$, is defined as the set of all (infinite) sequences satisfying the formula $\varphi$. We denote by $Sat(\varphi)$ the set of these sequences. We say that a property $\Pi$ is expressible (in temporal logic) if $\Pi = Sat(\varphi)$ for some temporal formula $\varphi$.

Next, consider the notion of a finitary property $\Phi \subseteq \Sigma^+$. To express such properties we use a past formula $p$. Let $\sigma \in \Sigma^+$ be a finite sequence of length $|\sigma| = n$ and $p$ a past formula. We say that $\sigma$ *end-satisfies* $p$, denoted by $\sigma \mathrel{=\!\!\!|} p$, if for some infinite extension $\sigma' \in \Sigma^\omega$, such that $\sigma \prec \sigma'$, $(\sigma', n-1) \models p$. That is, $p$ holds at the last position of $\sigma$ within $\sigma'$. It is not difficult to see that this definition is dependent only on the first $n$ positions of $\sigma'$, i.e., on $\sigma$. The finitary property represented by the past formula $p$ is defined to be the set of finite sequences that end-satisfy $p$. For a past formula $p$, denote by $esat(p)$, the finitary property defined by $p$, i.e., the set of finite sequences that end-satisfy $p$. We say that a finitary property $\Phi$ is expressible (in temporal logic) if it equals $esat(p)$ for some *past* formula $\varphi$.

For example, the finitary property $a^*b$ can be represented by the past formula $b \wedge \widetilde{\ominus}\boxminus a$, which claims that $b$ holds now and $a$ holds in all the preceding positions.

With these interpretations we can now show that the four language operators $A$, $E$, $R$, and $S$, when applied to expressible finitary properties, can be represented by the four future modalities $\Box$, $\Diamond$, $\Box\Diamond$, and $\Diamond\Box$. This is stated by the following claims

- $Sat(\Box p) \;=\; A\big(esat(p)\big)$
- $Sat(\Diamond p) \;=\; E\big(esat(p)\big)$
- $Sat(\Box\Diamond p) \;=\; R\big(esat(p)\big)$
- $Sat(\Diamond\Box p) \;=\; P\big(esat(p)\big)$

Consider, for example, the first clause. It is clear that $\sigma \in Sat(\Box p)$ iff $\sigma \models \Box p$ iff all prefixes of $\sigma$ end-satisfy $p$ iff all prefixes of $\sigma$ belong to $esat(p)$ iff $\sigma \in A\big(esat(p)\big)$.

Similar arguments justify the remaining three clauses.

In the following we will be interested in the closure of properties expressible by temporal formulae under the operations of union, intersection, and complementation. The following useful identities show that these operations on the properties can be translated to disjunction, conjunction, and negation of the formulae expressing the properties.

$$\begin{aligned} Sat(\varphi) \,\cup\, Sat(\psi) &= Sat(\varphi \vee \psi) \\ Sat(\varphi) \,\cap\, Sat(\psi) &= Sat(\varphi \wedge \psi) \\ \overline{Sat(\varphi)} &= Sat(\neg\varphi), \end{aligned}$$

where complementation is relative to $\Sigma^\omega$.

## Safety Formulae

We define a *safety formula* to be a formula of the form

$$\Box p,$$

for a past-formula $p$. Such a formula states that all positions in a computation satisfy $p$. Equivalently, all prefixes of a computation end-satisfy $p$.

A property that can be specified by a safety formula is called a *safety-specifiable* property. Clearly, a property specified by the safety formula $\Box p$ is a safety property, because it can be presented as

$$A\big(esat(p)\big).$$

We say that an arbitrary formula is *safety-equivalent* if it is equivalent to a safety property. Obviously, all safety-equivalent formulae specify safety properties.

Usually, safety formulae express *invariance* of some state property over all computations, or *precedence* constraints of the form: if events $e_1$ and $e_2$ happen then $e_1$ must precede $e_2$.

In the simpler cases, $p$ is a *state*-formula, and then the formula $\Box p$ specifies that all states in the computation satisfy $p$. An example of such a simple safety property is the formula

$$\Box(x \geq 0),$$

specifying that, in all states of the computation, the value of $x$ is nonnegative.

We will illustrate the utility of state invariances by presenting several typical examples.

- **Partial Correctness**

Let $P$ be a program whose terminal location is $\ell_t$. Let $\psi$ be an assertion specifying the post condition of the program, i.e., constraints on the final state of the program. For example, if $P$ computes the factorial of the input $x$, and places it in the output variable $z$, the post condition can be $\psi : (z = x)$. The program is defined to be partially correct, with respect to the post condition $\psi$ if every *terminating* computation must terminates in a state satisfying $\psi$. Nothing is implied about non-terminating computation. Partial correctness with respect to $\psi$ can be specified by the safety formula

$$\Box(at\_\ell_t \rightarrow \psi).$$

This formula states that it is invariantly true that if control is at the terminating location $\ell_t$, i.e., the program has terminated, then the post condition $\psi$ holds.

- **Mutual Exclusion**

Consider a program consisting of two processes $P_1$ and $P_2$ that need to coordinate their entry to critical sections in their code. The program for each process $P_i$ is usually partitioned into three sections: $N_i$, $T_i$, and $C_i$. The section $N_i$ represents the non-critical activity of the process, where no coordination is required. The section $T_i$ represents the *trying* section, where a process decides it needs to access its critical section, and engages in a protocol that will ensure eventual access. The section $C_i$ represents the critical section itself. The basic requirement of mutual exclusion algorithms is that it is never the case that both $P_1$ and $P_2$ execute their critical sections at the same time. This requirement can be expressed by the safety formula

$$\Box\neg(in\_C_1 \wedge in\_C_2),$$

where $in\_C_i$ is a control predicate expressing the fact that $P_i$ is currently executing within the section $C_i$.

The more general case of safety formulae of the form $\Box p$, where $p$ is not a state formula, is illustrated by the following examples:

- **Precedence**

The basic precedence formula has the form

$$\Box(q \rightarrow \diamondminus p).$$

It states that if $q$ ever occur, then it must be preceded by $p$. There are many applications and corresponding interpretations to this formula. For example, it can be interpreted as the property of *casual dependence* of $q$ upon $p$. That is, $q$ cannot happen unless it is caused by $p$. If $q$ is a response to the request $p$, then this property claims that the system does not repond spuriously, without being requested.

The same property can also be specified by the future formula $(\neg q)\mathcal{U}p$. While this is not a safety formula, it is equivalent to the safety formula given above, and therefore specifies precisely the same property.

- **FIFO ordering**

Assume that $q$ represents a response to a request $p$, and $q'$ represents a similar but disjoint response to the request $p'$. For example, the two may represent similar responses to different customers. The following safety formula states that the order of the responses matches the order of the requests.

$$\Box\big((q \wedge \diamondminus q') \rightarrow \diamondminus(p \wedge \diamondminus p')\big)$$

Note that this formula does not guarantee any responses, but it claims that if they appear, they appear in the right order.

### Closure of Properties Expressible by Safety Formulae

The class of properties expressible by safety formulae is closed under the positive boolean operations, i.e., intersection and union. As stated before, it suffices to show that if $\varphi$ and $\psi$ are safety formulae then both $\varphi \wedge \psi$ and $\varphi \vee \psi$ are equivalent to safety formulae. That is, it suffices to show that the class of safety-equivalent formulae is closed under conjunction and disjunction.

To see this we present the following equivalences for the conjunction and disjunction of safety formulae:

$$\begin{array}{lcl}(\Box p \ \wedge \ \Box q) & \sim & \Box(p \ \wedge \ q) \\ (\Box p \ \vee \ \Box q) & \sim & \Box(\boxminus p \ \vee \boxminus q).\end{array}$$

The left-hand side of the second equivalence states, for a computation $\sigma$, that either all positions in $\sigma$ satisfy $p$ or all positions in $\sigma$ satisfy $q$. The right-hand side states that for *each* position $i$, either all positions $j \le i$ satisfy $p$, or all positions $j \le i$ satisfy $q$. To see that the right-hand side implies the left-hand side, we consider two cases. If all positions in $\sigma$ satisfy both $p$ and $q$ then, clearly, the left-hand side follows. If for some $j$, $(\sigma, j) \not\models p$, then the only way the right-hand side can hold is by having for all $i \ge j$, $(\sigma, i) \models \boxminus q$, from which $\Box q$ follows.

Since the right-hand side of both equivalences is a safety formula (under the assumption that $p$ and $q$ are past-formulae), this establishes the closure of properties expressible by safety formulae under intersection and union.

An important formula is the formula of *conditional safety*, in which a property expressed by $\Box q$ is conditional on a state-formula $p$ holding at the first state of the computation. This formula has the form

$$p \rightarrow \Box q.$$

While not being a safety formula itself, this formula is safety-equivalent due to the equivalence

$$(p \rightarrow \Box q) \;\sim\; \Big(\Box\big(\diamondsuit(p \wedge \mathit{first}) \rightarrow q\big)\Big).$$

The formula on the right states that at each position $j$, if $j$ has been preceded by some position $i \leq j$ that satisfies $p$ and is also *first* (forcing $i = 0$), then $q$ should hold at $j$.

- **Full Partial Correctness**

In specifying terminating programs, one usually specifies a *precondition* $\varphi$, in addition to the *postcondition* $\psi$. The role of the precondition is to constrain the inputs for which the program is expected to produce the right result. For example, for the factorial computing program, a natural precondition is: $\varphi : (x \geq 0)$, claiming that the program is expected to produce a correct result only if we start it with a non-negative input. The partial correctness of a program $P$ with respect to both the precondition $GF$ and the postcondition $GP$ can be specified by the conditional safety formula

$$\varphi \rightarrow \Box(at\_\ell_t \rightarrow \psi).$$

## Guarantee Formulae

A *guarantee formula* is a formula of the form

$$\diamondsuit p,$$

for some past-formula $p$. Such a formula states that at least one position in a computation satisfies $p$.

A property that can be specified by a guarantee formula is called a *guarantee-specifiable property*. Clearly, any property that can be specified by the guarantee formula $\diamondsuit p$ is a guarantee property, since it can be presented as

$$E\big(esat(p)\big).$$

An arbitrary formula is defined to be *guarantee-equivalent* if it is equivalent to a guarantee formula.

Usually, guarantee formulae ensure that some event *eventually* happens. They guarantee that the event happens at least once, but cannot promise any repetitions of the event. Therefore, they are mainly used to ensure events that happen *once* in the lifetime of a program execution, such as termination.

Obviously, a formula specifies a guarantee-specifiable property iff it is equivalent to some guarantee formula.

An example of the simple case, in which $p$ is a state-formula, is the formula

$$\diamondsuit(terminal)$$

specifying that some state of the computation is terminal. Clearly if all computations of a given program satisfy this formula, the program is terminating. Instead of using the abstract predicate *terminal*, we can use the more concrete formula $\diamondsuit(at\_\ell_t)$, claiming that all computations eventually reach the terminal location $\ell_t$.

We observe that the *conditional guarantee* formula

$$p \rightarrow \diamondsuit q,$$

while not being a guarantee formula, still specifies guarantee-specifiable properties. This is because it is equivalent to the guarantee formula

$$\diamondsuit(first \wedge p \rightarrow q).$$

This formula states that eventually we reach a position such that if, looking back towards the origin, we detect that $p$ held at the initial position, then $q$ holds now.

- **Total Correctness**

Consider a program $P$ associated with a precondition $\varphi$ and a postcondition $\psi$. The program $P$ is said to be *totally correct* with respect to $(\varphi, \psi)$ if *all* computations starting at a $\varphi$-state terminate at a$\psi$-state. This property can be expressed by the conditional guarantee formula

$$\varphi \rightarrow \diamondsuit(at\_\ell_t \wedge \psi).$$

### Closure of Guarantee-Specifiable Properties

Many features of the guarantee-specifiable class of properties can be obtained by the duality relation between the safety-specifiable and guarantee-specifiable classes. This duality is based on the equivalence

$$\neg\Box p \;\sim\; \diamondsuit(\neg p).$$

From this equivalence we can immediately conclude that $\Pi$ is a guarantee-specifiable property iff the complementary property $\overline{\Pi}$ (i.e., the set of all computations *not* in $\Pi$) is a safety-specifiable property.

In principle we could justify the closure properties of the guarantee-specifiable class, using duality and the corresponding closure properties of the safety-specifiable class. However, we prefer to give an independent justification.

Similarly to the class of safety-equivalent formulae, the class of guarantee-equivalent formulae is closed under the positive boolean operations of disjunction and conjunction.

This can be shown using the following equivalence and equivalence:

$$\begin{aligned}(\Diamond p \ \vee \ \Diamond q) &\sim \Diamond[p \ \vee \ q] \\ (\Diamond p \ \wedge \ \Diamond q) &\sim \Diamond[\diamondminus p \ \wedge \ \diamondminus q].\end{aligned}$$

The second equivalence claims that a computation $\sigma$ contains both a $p$-position (a position satisfying $p$) and a $q$-position iff it has a position $i$ such that there exist a $q$-position $j \leq i$, and a $p$-position $k \leq i$, preceding $i$.

The class of guarantee-equivalent formulae is not closed under negation. On the other hand, the negation of a guarantee formula is equivalent to a safety formula. Similarly, the negation of a safety formula is equivalent to a guarantee formula.

This is due to the following two equivalences:

$$\begin{aligned}(\neg\Diamond p) &\sim \Box(\neg p) \\ (\neg\Box p) &\sim \Diamond(\neg p).\end{aligned}$$

We say that the classes of guarantee-equivalent and safety-equivalent formulae are *dual*; as each can be obtained by the negation of the other.

## Obligation Formulae

Some properties cannot be expressed by either safety or guarantee formulae alone, and must be expressed as a boolean combination of such formulae. We therefore consider the class of such properties.

A *simple obligation* is a formula of the form

$$\Box p \ \vee \ \Diamond q,$$

where $p$ and $q$ are past formulae. This formula states that either $p$ holds at all positions of a computation or $q$ holds at some position.

A property that can be specified by a simple obligation formula is called a *simple obligation specifiable property*. Clearly, any property that can be specified by the simple obligation formula $\Box p \vee \Diamond q$ is a simple obligation property, since it can be presented as

$$A\big(esat(p)\big) \ \cup \ E\big(esat(q)\big).$$

An obviously equivalent form for a simple obligation formula is

$$\Diamond r \ \rightarrow \ \Diamond q,$$

which states that if some position satisfies $r$ then some position (possibly the same) satisfies $q$.

- **Exceptions**

A typical example of properties that are naturally specified by obligation formulae is that of exceptional occurrences. Assume a program that in the normal course of its behavior is not supposed to terminate but to maintain some regular activity. However, in the case of the occurrence of some exceptional event $p$, it is supposed to take some exceptional action $q$ and to terminate. Specifying this behavior can be done by the guarantee formula

$$\Diamond p \rightarrow \Diamond(q \wedge \diamondminus p).$$

Note that this formula also guarantees that $q$ happens only *after* some occurrence of $p$.

**General Obligation Formulae**

The class of properties specifiable by simple obligation formulae is closed under union. To see this, we observe the trivial equivalence

$$\begin{aligned}&\big[(\Box p_1 \vee \Diamond q_1) \ \vee \ (\Box p_2 \vee \Diamond q_2)\big] \ \sim \\ &\qquad \big[(\Box p_1 \vee \Box p_2) \ \vee \ (\Diamond q_1 \vee \Diamond q_2)\big].\end{aligned}$$

Using the closure of both the safety- and guarantee-equivalent formulae under disjunction, this leads to an equivalent simple obligation formula. However, the class of properties specifiable by simple obligation formulae is *not* closed under intersection. This implies that by taking conjunctions of simple obligation formulae we obtain a more powerful class.

We therefore define an *obligation formula* to be a formula of the form

$$\bigwedge_{i=1}^{n}\big[\Box p_i \ \vee \ \Diamond q_i\big].$$

Correspondingly, a property specifiable by such a formula is called an *obligation property*. A formula that is equivalent to an obligation formula is called *obligation-equivalent*

This class is the largest class that can be obtained by taking finite boolean combinations of safety and guarantee formulae.

**Claim**

Every boolean combination of safety and guarantee formulae is equivalent to an obligation formula.

To see this, consider an arbitrary boolean combination of safety and guarantee formulae. First we push all negations into the past-formulae, changing $\wedge$ into $\vee$, $\Box$ into $\Diamond$, and vice-versa. Next we bring the formula into a conjunctive normal form:

$$\bigwedge_{i=1}^{n}\big[\Box p_1^i \ \vee \ \ldots \ \vee \ \Box p_{k_i}^i \ \vee \ \Diamond q_1^i \ \vee \ \ldots \ \vee \ \Diamond q_{m_i}^i\big].$$

We then use the closure properties of the safety and guarantee formulae to collapse all of $\Box p_1^i \vee \ldots \vee \Box p_{k_i}^i$ into a single safety formula, and $\Diamond q_1^i \vee \ldots \vee \Diamond q_{m_i}^i$ into a single guarantee formula.

This claim also implies that the class of obligation-equivalent formulae is closed under all boolean operations.

**Inclusion**

The class of obligation-specifiable properties strictly contains the classes of safety specifiable and guarantee specifiable properties. In fact, the property described by $\Box p \vee \Diamond q$ for propositions $p$ and $q$ cannot be specified by either safety or guarantee formulae.

The class of obligation-specifiable properties forms an entire infinite strict hierarchy. The class of properties expressible by a conjunction of $n+1$ simple obligation formulae strictly contains the class corresponding to a conjunction of only $n$ simple obligation formulae.

## Recurrence Formulae

A *recurrence formula* is a formula of the form:

$$\Box\Diamond p,$$

for some past-formula $p$. It states that infinitely many positions in the computation satisfy $p$.

A property that can be specified by a recurrence formula is called a *recurrence-specifiable property*.

Clearly, a property specifiable by the recurrence formula $\Box\Diamond p$ is a recurrence property, since it can be presented as

$$R\big(esat(p)\big).$$

A formula that is equivalent to a recurrence formula is called *recurrence-equivalent*. One of the most important recurrence-equivalent formulae is the following formula, called a *response formula*

$$\Box(p \rightarrow \Diamond q).$$

The fact that this formula is recurrence equivalent is established by the equivalence

$$\Box(p \rightarrow \Diamond q) \quad \sim \quad \Box\Diamond\big((\neg p) \mathcal{U} q\big).$$

The formula on the right states that there exists infinitely many positions in which there is no pending request, i.e., a request that has not been followed by a response.

Usually, recurrence properties ensure that some event happens infinitely many times. When specified by response formulae, they can express the property of *responsiveness* of a system, stating that every stimulus has a recurrence.

- **Accessibility**

Consider again a program for solving the mutual exclusion problem. As already mentioned in the introduction, the safety property that disallows the two processes to co-reside in their critical sections is only part of the specification. It can easily be implemented by a program that does not allow any of the processes to ever access its critical section. To exclude such spurious solutions we must add to the specification the requirement that each process interested in entering the critical section will eventually succeed. This property can be specified by the response formula

$$\Box(in\_T_i \rightarrow \Diamond in\_C_i).$$

This formula requires that whenever one of the processes is observed to occupy the trying section $T_i$, then eventually it will be observed in the critical section $C_i$.

**Closure of Recurrence-Specifiable Properties**

The class of properties expressible by recurrence formulae is closed under the positive boolean operations.

This is shown by the following equivalences:

$$\begin{aligned} [\Box\Diamond p \ \vee \ \Box\Diamond q] \quad &\sim \quad \Box\Diamond(p \ \vee \ q) \\ [\Box\Diamond p \ \wedge \ \Box\Diamond q] \quad &\sim \quad \Box\Diamond\Big(q \ \wedge \ \ominus\big((\neg q)\,\mathcal{S} p\big)\Big). \end{aligned}$$

The past formula $q \wedge \ominus\big((\neg q)\,\mathcal{S} p\big)$ expresses precisely the *minex* operator applied to the finitary properties $esat(p)$ and $esat(q)$, that is,

$$esat\Big(q \ \wedge \ \ominus\big((\neg q)\,\mathcal{S} p\big)\Big) \ = \ minex\big(esat(p), \ esat(q)\big).$$

To see this, let $\sigma$ be a finite sequence that end-satisfies $q \wedge \ominus\big((\neg q)\,\mathcal{S} p\big)$. By definition its last position satisfies $q$, and hence $\sigma \in esat(q)$. The formula $\ominus\big((\neg q)\,\mathcal{S} p\big)$, holding at the last position of $\sigma$, requires that there exists some proper prefix $\sigma' \prec \sigma$, such that $\sigma'$ end-satisfies $p$, and hence $\sigma' \in esat(p)$, and for every $\sigma''$, $\sigma' \prec \sigma'' \prec \sigma$, $\sigma''$ does *not* end-satisfy $q$, i.e., $\sigma'' \notin esat(q)$. It clearly follows that $\sigma \in minex\big(esat(p), \ esat(q)\big)$.

**Inclusion of the Lower Classes**

All safety and guarantee formulae can be shown to be special cases of recurrence formulae. Thus, the class of recurrence-specifiable properties contains the classes of safety-specifiable and guarantee-specifiable properties.

This containment is supported by the following two equivalences:

$$\begin{aligned}\Box p &\sim \Box\Diamond(\boxminus p)\\ \Diamond p &\sim \Box\Diamond(\diamondminus p).\end{aligned}$$

The second equivalence, for example, states that a computation $\sigma$ has a $p$-position iff there are infinitely many positions in whose past there is a $p$-position.

The containment of both classes is strict. This means that there is a recurrence property, which cannot be expressed by either a safety or a guarantee formula. In fact, the formula $\Box\Diamond p$, for a state formula $p$, cannot even be expressed by any finite boolean combination of safety and guarantee formulae.

### Expressing Weak Fairness

One of the important properties belonging to the recurrence-specifiable class is that of *weak fairness*. The representation of concurrent programs as fair transition systems (see for example [MP83]) associates a *weak fairness* requirement (also called *justice*) with each transition $\tau$ in the system. This requirement can be formulated as

> It is not the case that, from a certain point on, the transition $\tau$ is continually enabled but never taken.

Thus, any computation satisfying this requirement must have infinitely many positions at which either $\tau$ is disabled or $\tau$ is taken. This can be expressed by the recurrence formula

$$\Box\Diamond\big[\neg En(\tau) \ \vee \ taken(\tau)\big].$$

We assume the existence of the state predicates $En(\tau)$ and $taken(\tau)$, which test whether the transition $\tau$ is enabled or taken at a given state.

## Persistence Formulae

A *persistence formula* is a formula of the form

$$\Diamond\Box p,$$

for some past-formula $p$. The formula states that all but finitely many positions (all positions from a certain point on) in the computation satisfy $p$.

A property that can be specified by a persistence formula is called a *persistence-specifiable property*. Clearly, any property that can be specified by the persistence formula $\Diamond\Box p$ is a persistence property, since it can be presented as

$$P\big(esat(p)\big).$$

A formula that is equivalent to a persistence formula is called *persistence-equivalent*.

Usually, persistence formulae are used to describe the eventual stabilization of some state or past property of the system. They allow an arbitrary delay until the stabilization occurs, but require that once it occurs it is continuously maintained.

For example, $p$ may represent a certain stimulus to the system, and it is required that following an occurrence of $p$, the system will eventually stabilize by continuously maintaining $q$. This requirement may be specified by the *conditional persistence* formula

$$\Box(p \rightarrow \Diamond\Box q),$$

which is persistence-equivalent, since it is equivalent to the persistence formula

$$\Diamond\Box(\diamondminus p \rightarrow q).$$

The latter formula states that all the states, from a certain point on, satisfy the requirement that if $p$ has already occurred then $q$ currently holds. Note that this also covers the case that $p$ never occurs, and hen nothing is implied about $q$.

### Closure of Persistence-Specifiable Properties

The class of properties that can be expressed by persistence formulae is closed under the positive boolean operations.

This is shown by the following equivalences that can also be derived by duality from the corresponding equivalences for the recurrence case:

$$\begin{aligned}(\Diamond\Box p \ \wedge \ \Diamond\Box q) &\sim \Diamond\Box(p \ \wedge \ q)\\ (\Diamond\Box p \ \vee \ \Diamond\Box q) &\sim \Diamond\Box\Big( q \vee \\ &\qquad \ominus\big(p\,\mathcal{S}\big(p \ \wedge(\neg q)\big)\big)\Big).\end{aligned}$$

To see the validity of the second equivalence, we will show first that the left-hand side implies the right-hand side.

Obviously, $\Diamond\Box q$ implies the right-hand side. If $\Diamond\Box p$ is true and $\Diamond\Box q$ is not, let $i$ be the position beyond which $p$ is true, and $j \geq i$ some position at which $q$ is false (by $\Diamond\Box q$ being false there are infinitely many such positions). It is easy to see that for every position $k \geq j$, $(\sigma, k) \models \ominus\big(p\,\mathcal{S}\big(p \wedge (\neg q)\big)\big)$.

Next, we show that the right-hand side implies the left-hand side. Again, we consider two cases. If $\Diamond\Box q$ holds, then obviously the left-hand side follows. In the other case, there are infinitely many $\neg q$-positions. Let $i$ be the position beyond which $\psi$: $q \vee \ominus\big(p\,\mathcal{S}(p \wedge (\neg q))\big)$ continuously holds. Consider an arbitrary position $j \geq i$, and let $k > j$ be the first $\neg q$-position to the strict right of $j$. Since $\psi$ holds at $k$ and $q$ does not, it follows that $\ominus\big(p\,\mathcal{S}(p \wedge (\neg q))\big)$ must hold at $k$. This means that $p$ must extend from $k-1$ to the left up to, and including, the first $\neg q$-position to the left of $k$. Since there is no $\neg q$-position between $j$ and $k$, $p$ must also hold at $j$. Since $j$ is an arbitrarily chosen position to the right of $i$, it follows that $\Box p$ holds at $i$, and hence $\Diamond\Box p$ holds at position 0.

The classes of properties specifiable by persistence and recurrence formulae are *dual*. This means that the complement of a property in one of the classes belongs to the other. This is supported by the two equivalences:

$$\begin{array}{rcl} \neg(\Box\Diamond p) & \sim & \Diamond\Box(\neg p) \\ \neg(\Diamond\Box p) & \sim & \Box\Diamond(\neg p). \end{array}$$

This duality can be used for easy transfer of results holding for one class into the other class. For example, all the closure and inclusion properties, of the persistence-specifiable class, as well as the proofs about the extended persistence formulae, can be derived from the corresponding properties and proofs of the recurrence-specifiable class.

**Inclusion of the Lower Classes**

All safety and guarantee formula are special cases of the persistence formula. Thus, the class of persistence-specifiable properties contains the classes of properties specifiable by safety and guarantee formulae.

This containment is supported by the following two equivalences:

$$\begin{array}{rcl} \Box p & \sim & \Diamond\Box(\boxminus p) \\ \Diamond p & \sim & \Diamond\Box(\diamondminus p). \end{array}$$

The second equivalence, for example, states that a computation $\sigma$ has a $p$-position iff all positions, from a certain point on, have $p$ in their past.

The containment of both classes is strict. This is shown by the property $\Diamond\Box p$ for a state-formula $p$, that cannot be expressed by either a safety or a guarantee formula. In fact, it cannot be expressed by any finite boolean combination of safety and guarantee formulae.

## Simple Reactivity Formulae

A *simple reactivity formula* is a formula formed by a disjunction of a recurrence formula and a persistence formula

$$\Box\Diamond p \ \vee \ \Diamond\Box q.$$

This formula states that either the computation contains infinitely many $p$-positions, or all but finitely many of its positions are $q$-positions.

A property that can be specified by a simple reactivity formula is called a *simple reactivity-specified property*. Clearly, any property that can be specified by the simple reactivity formula $\Box\Diamond p \vee \Diamond\Box q$ is a simple reactivity property, since it can be presented as

$$R\big(esat(p)\big) \ \cup \ P\big(esat(q)\big).$$

In many cases we specify such properties by a formula of the form

$$\Box\Diamond r \ \rightarrow \ \Box\Diamond p,$$

which is obviously equivalent to a simple reactivity formula.

This formula states that if the computation contains infinitely many $r$-positions it must also contain infinitely many $p$-positions. It is used to describe responsiveness of a more complicated type, which is not based on one-to-one correspondence between stimulus to response. It is only when we have infinitely many stimuli that we must respond by infinitely many responses. This is a convenient abstraction to a situation in which we want to commit the system to eventually respond, but not specify a bound on how many stimuli may happen before the eventual response.

**The Different types of Responsiveness — A Summary**

So far, we have encountered several types of responsiveness, which can be specified by formulae belonging to the different classes. Let us review these different versions. As usual, assume that $p$ represents a stimulus, to which the system responds by producing $q$.

- The guarantee-equivalent formula

  $$p \rightarrow \Diamond q,$$

  ensures that if $p$ is true *initially* then $q$ will eventually happen.

- The obligation-equivalent formula

  $$\Diamond p \rightarrow \Diamond(q \wedge \diamondminus p),$$

  ensures that if $p$ happens at least once, then its earliest occurrence will be followed by at least *one* occurrence of $q$.

- The recurrence-equivalent formula

  $$\Box(p \rightarrow \Diamond q),$$

  ensures that *every* occurrence of $p$ is followed by an occurrence of $q$.

- The persistence-equivalent formula

$$p \rightarrow \Diamond\Box q,$$

ensures that an occurrence of $p$ will be eventually followed by a continuous maintenance of $q$.

- The reactivity-equivalent formula

$$\Box\Diamond p \rightarrow \Box\Diamond q,$$

ensures that infinitely many occurrences of $p$ are responded to by infinitely many occurrences of $q$.

The type of responsiveness represented by simple reactivity formulae allows the program $P$ to ignore finitely many requests but not infinitely many of them. This description should not be taken too literally, in the sense that no implementation of this requirement can be based on the idea of "let us wait first and see whether there are going to be infinitely many $x = 1$ events or only finitely many of them." Any reasonable implementation of such a requirement must sincerely attempt to respond to all requests, but the liberal specification tolerates failures to respond in the case of only finitely many requests.

The class of properties specifiable by simple reactivity formulae is closed under union. This is due to the equivalences

$$\big[(\Box\Diamond p_1 \;\vee\; \Diamond\Box q_1) \;\vee\; (\Box\Diamond p_2 \;\vee\; \Diamond\Box q_2)\big] \sim \big[(\Box\Diamond p_1 \;\vee\; \Box\Diamond p_2) \;\vee\; (\Diamond\Box q_1 \;\vee\; \Diamond\Box q_2)\big]$$

and the closure of the recurrence and persistence-specifiable classes under union.

However, the reactivity-specifiable class is, in general, not closed under intersections or complementations.

Obviously, the class of properties specifiable by simple reactivity formulae contains the classes of properties specifiable by recurrence and persistence formulae, and hence also the classes specifiable by safety, guarantee and obligation formulae. This containment is strict since the property specifiable by $\Box\Diamond p \vee \Diamond\Box q$ cannot be expressed by any formula belonging to a lower class.

### Expressing Strong Fairness

Simple reactivity formulae can express the requirement of strong fairness associated with special transitions of a fair transition system. Typically, we associate strong fairness requirements with transitions that correspond to communication or synchronization statements in the program, such as statements dealing with semaphores.

The strong fairness requirement associated with a transition $\tau$ demands that

> It is not the case that $\tau$ is enabled infinitely many times but taken only finitely many times.

Equivalently, this requirement demands that if the transition $\tau$ is enabled infinitely many times in a computation $\sigma$, then it must be taken infinitely many times. This can be expressed by the simple reactivity (-equivalent) formula

$$\Box\Diamond\, En(\tau) \;\rightarrow\; \Box\Diamond\, taken(\tau).$$

## General Reactivity Properties

Richer classes of properties can be expressed by conjunctions of simple reactivity formulae of the form

$$\bigwedge_{i=1}^{n} \Big[\Box\Diamond p_i \;\vee\; \Diamond\Box q_i\Big].$$

Since, in general, the conjunction of two simple reactivity formulae is not equivalent to any simple reactivity formula, taking such conjunctions leads to a stronger expressive power.

We call such formulae *reactivity formulae*, and the properties they specify *reactivity-specifiable properties*.

The class of reactivity-specifiable properties is the maximal class we need ever consider. This is due to the following normal form theorem:

> **Theorem** (reactivity)
>
> Every temporal formula is equivalent to a reactivity formula.

The proof of this theorem is based on a translation between future and past temporal formulae. A detailed proof of this theorem is beyond the scope of this paper.

A natural example of a property specifiable by a reactivity formula is the total statement of fairness for a fair transition system. Since each individual fairness requirement is expressible by a simple reactivity formula (recurrence formula if it is a weak fairness requirement), the statement that *all* fairness requirements hold is expressible as the conjunction of several simple reactivity formulae.

Our approach to specification of programs is inherently conjunctive. This means that a specification is presented as a conjunction of requirements, expressed by temporal formulae, all of which should be valid over the program. In verifying that a specification is valid over a given program, we can verify the validity of each requirement separately. Therefore, the fact that one of the requirements is a conjunction by itself, rather than a simple reactivity formula, does not greatly complicate or simplify the situation. Hence, in the context of a full specification, which is always a conjunction, we may assume each requirement to be at most a simple reactivity formula.

The family of properties specifiable by reactivity formulae forms an infinite hierarchy by itself. Level $k$ of the hierarchy, for $k > 0$, consists of all the properties that can be specified by a conjunction

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

for some $n \leq k$. This hierarchy is strict, since the conjunction

$$\bigwedge_{i=1}^{n+1} [\Box\Diamond p_i \;\vee\; \Diamond\Box q_i]$$

with $p_i$, $q_i$, $i = 1, \ldots, n+1$, being uninterpreted propositions is not equivalent to any conjunction of $n$ or fewer simple reactivity formulae.

## Relating the Syntactic and Semantic Classifications

As we will see, the syntactic hierarchy of properties, based on their expression by particular form of formulae, is identical to the semantic hierarchy, based on the construction of infinitary properties by applying the operators $A$, $E$, $R$, and $S$, to finitary properties. When introducing the class corresponding to each type of formula $\kappa$, where

$$\kappa \in \left\{ \begin{array}{lll} \textit{safety}, & \textit{guarantee}, & \textit{obligation}, \\ \textit{recurrence}, & \textit{persistence}, & \textit{reactivity} \end{array} \right\}$$

we immediately showed that any $\kappa$-specifiable property is a $\kappa$ property according to the semantic classification. For example, as soon as we defined the notion of a safety formula, we have shown that any property specifiable by a safety formula is a safety property according to the semantic classification.

We are now ready to consider the other direction. Suppose $\Pi$ is an infinitary property, that is known to be specifiable by a temporal formula, and is *also* known to be a $\kappa$-property. Can we conclude that it is a $\kappa$-specifiable property? We have already answered this question positively for the case $\kappa = \textit{reactivity}$. This is due to the previous theorem that stated that any temporal formula is equivalent to a reactivity formula. The next theorem answers this question positively for the other classes as well.

> **Theorem**
>
> Every infinitary property of type $\kappa$ that is specifiable by a temporal formula, is specifiable by a formula of type $\kappa$ (every specifiable $\kappa$-property is a $\kappa$-specifiable property).

The correspondence between the syntactic and semantic classification includes also the subhierarchies within the obligation and reactivity classes. For example, if a property is specifiable, and can also be presented as the intersection $\bigcap_{i=1}^{n}\big(A(\Phi_i') \cup E(\Phi_i'')\big)$; then it can also be specified by a formula of the form $\bigwedge_{i=1}^{n}(\Box p_i \vee \Diamond q_i)$ for some past formulae $p_i$ and $q_i$, $i = 1, \ldots, n$.

It is beyond the scope of this paper to give a proof of this theorem. We refer the reader to [Zuc86], where some of these issues are discussed.

## The Syntactic Characterization of Liveness

Similarly to our previous efforts to give a syntactic characterization to the semantic hierarchy, we provide a syntactic characterization to the class of *liveness* properties.

We define a *liveness formula* to be a formula of the form

$$\Diamond\left(\bigvee_{i=1}^{n}(p_i \;\wedge\; \Diamond q_i)\right),$$

where $q_i$, $i = 1, \ldots, n$, are satisfiable *future* formulae, and $p_i$, $i = 1, \ldots, n$, are *past* formulae, such that $\Box\big(\bigvee_{i=1}^{n} p_i\big)$ is a valid formula.

It is not difficult to see that any infinitary property $\Pi$ specifiable by a liveness formula is a liveness property. To see this, let $\sigma \in \Sigma^{+}$ be any finite sequence of length $|\sigma| = m$. By the requirement that $\Box\big(\bigvee_{i=1}^{n} p_i\big)$ is valid, $\sigma$ end-satisfies the disjunction $\bigvee_{i=1}^{n} p_i$, which means that it end-satisfies one of the disjuncts, say $p_j$. Since $q_j$ is satisfiable there exists an infinite sequence $\sigma'$ satisfying $q_j$. Consider the sequence $\sigma'' = \sigma \cdot \sigma'$, obtained by concatenating $\sigma'$ to the end of $\sigma$. It is obvious that position $m-1$ in $\sigma''$ satisfies $p_j$, and position $m$ satisfies $q_j$. It follows that position $m-1$ satisfies $p_j \wedge \Diamond q_j$, and therefore $\sigma''$ satisfies $\Diamond\big(\bigvee_{i=1}^{n}(p_i \wedge \Diamond q_i)\big)$.

Consider the formula $(p \;\rightarrow\; \Diamond\Box q) \wedge \big((\neg p) \;\rightarrow\; \Diamond\Box(\neg q)\big)$, where $p$ and $q$ are propositions. This formula specifies a property $\Pi$ which is a liveness property but not a uniform liveness property. Indeed this formula is equivalent to the liveness formula

$$\diamond\Big[\Big(\diamondminus(\mathit{first}\ \wedge\ p)\ \wedge\ \diamond\Box q\Big)\ \vee$$
$$\Big(\diamondminus(\mathit{first}\ \wedge\ (\neg p))\ \wedge\ \diamond\Box(\neg q)\Big)\Big].$$

Clearly, the formula $\Box\big(\diamondminus(\mathit{first}\wedge p)\vee\diamondminus\big(\mathit{first}\wedge(\neg p)\big)\big)$ is valid.

The following theorem establishes the connection between the semantic and syntactic characterization of liveness.

**Theorem**

A property $\Pi$ that is specifiable in temporal logic, is a liveness property iff it is specifiable by a liveness formula.

Again, we provide no proof of this theorem.

Since our main interest is in properties that can be specified by temporal logic, we will drop in the future the *specifiable* qualifier. Thus, when we will talk about safety properties, we mean specifiable safety properties, and referring to liveness properties, we mean specifiable liveness properties.

An alternative syntactic characterization of liveness is given by the formula

$$\diamond\Big(\bigwedge_{i=1}^{n}(p_i\ \rightarrow\ \diamond q_i)\Big),$$

where $q_i$, $i=1,\ldots,n$, are satisfiable future formulae, and $p_i$, $i=1,\ldots,n$, are past formulae, such that, for every $i\neq j$, $\Box\neg(p_i\wedge p_j)$ is a valid formula.

# 5 The Automata View

An alternative formalism for specifying temporal properties is that of finite-state predicate automaton (see [AS89], [MP87]). 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.
- $T=\{t(q_i,q_j)\mid q_i,q_j\in Q\}$ – 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 $q_j$. We assume that each $t(q_i,q_j)$ is either syntactically identical to the constant F, 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 $\sigma$ if:

1. The first state of the run, $q_0$, 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 $\inf(r)\cap R\neq\phi$ or $\inf(r)\subseteq P$. The automaton $\mathcal{M}$ *accepts* the computation $\sigma$ if there exists a run of $\mathcal{M}$ over $\sigma$ which is accepting. This definition of acceptance has been introduced by Streett ([Str82]).

An alternative definition, given in [MP87] is that *all* runs of $A$ over $\sigma$ are accepting.

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

$$\bigvee_{q'\in Q} t(q,q')=\mathrm{T}.$$

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 $t(q,q'')$ true at the same time.

In this paper we restrict our attention to complete deterministic automata. Deterministic automata have exactly one run $r$ corresponding to each input computation $\sigma$, and hence the definition of acceptance in [MP87] 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 there is no transition from $q\in B$, to $q'\in G$, i.e., for every $q\in B, q'\in G$, $t(q,q')\equiv\mathrm{F}$. That is, the automaton cannot move from a bad state ($q\in B$) to a good state ($q'\in G$).
- A *guarantee* automaton is such that there is no transition from $q\in G$ to $q'\in B$.

- A *simple obligation* automaton is such that:
  - There is no transition from $q \notin P$ to $q' \in P$.
  - There is no transition from $q \in R$ to $q' \notin R$.

  The above definition implies that once a run exits $P$, it can never reenter $P$ again, and once it enters $R$, it can never get out. This can be generalized to general obligation as follows:

- A (general) *obligation* automaton (of degree $k$) is an automaton, in which each state $q \in Q$ has a rank $\rho(q)$, $0 \leq \rho(q) \leq k$, such that:
  - There is a transition from $q$ to $q'$, i.e., $t(q, q') \neq \mathrm{F}$, only if $\rho(q) \leq \rho(q')$.
  - There is a transition from $q \in B$ to $q' \in G$ only if $\rho(q) < \rho(q')$.
  - There is no transition from a state $q \in G$ of rank $k$ to a state $q' \in B$.

  This definition leads to the fact that a run can move from $B$ to $G$ (equivalently, move from $G$ into $B$), at most $k$ times. It is easy to see that the case of $k = 1$ corresponds to the definition of a simple obligation automaton, with $P$ being the set of $G$-states with rank 0, and $R$ being the set of $G$-states with rank 1.

- A *recurrence* automaton is such that $P = \phi$.
- A *persistence* automaton is such that $R = \phi$.
- A *simple reactivity* automaton is an unrestricted automaton of the above type.

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

In order to attain expressive power comparable to (and even exceeding, see [Wol83]) that of temporal logic we have to consider a more general type of automaton.

We define a *Streett Predicate Automaton* to be a structure

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

where $Q, q_0$, and $T$ are as defined above, and $L$ is a finite list of pairs of acceptance sets.

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

A run $r$ of a Streett automaton is accepting if for *each* $i = 1, \ldots, k$, either $\inf(r) \cap R_i \neq \phi$ or $\inf(r) \subset P_i$. The notions of computations accepted by such automaton and the properties specified by it are similar to the simpler case. This type of automaton has been studied by Streett in [Str82], and is the dual of Rabin's automaton ([Rab72]).

Obviously, all the preceding types of automata are special cases of a Streett automaton with $k = 1$. We associate a general Streett automaton with the class of (general) reactivity properties.

An infinitary property $\Pi \subseteq \Sigma^{\omega}$ is defined to be *specifiable by automata*, if there exists a Streett automaton $\mathcal{M}$, which accepts an infinite sequence $\sigma$ iff $\sigma \in \Pi$. The following proposition relates the syntactic characterization of the different types of automata to the semantic characterization of the properties they specify.

**Proposition 5.1** *A property* $\Pi$*, that is specifiable by automata, is a* $\kappa$*-property iff it is specifiable by a* $\kappa$*-automaton, where* $\kappa \in$ *{safety, guarantee, obligation, recurrence, persistence, reactivity}.*

For most of these types, this proposition has been proved in [Lan69], with some minor differences in the definitions of a safety and guarantee automata. The case of reactivity, and in fact the complete hierarchy above, has been solved in [Wag79].

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 a $\kappa$-automaton specifies a $\kappa$-property. Let $\mathcal{M}$ be a $\kappa$-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 $\sigma$ terminates ($\sigma$ is finite) at $q$.

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

Obviously, an infinite $\sigma$ 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 $\sigma$ contains infinitely many prefixes in $\Pi(q)$ for $q \in R$, or that all but finitely many prefixes of $\sigma$ are in $\Pi(q)$ for some $q \in P$. It follows that

$$\Pi_{\mathcal{M}} = R(\bigcup_{q \in R} \Pi(q)) \cup S(\bigcup_{q \in P} \Pi(q)).$$

Consequently, every property specifiable by a single automaton is a reactivity property. However, as we will show for the special cases of $\kappa$-automata, this expression can be further simplified.

■ For a safety automaton, it is clear that no finite prefix of an acceptable computation can be in $\Pi_B = \bigcup_{q \in B} \Pi(q)$. 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

$$\Pi_{\mathcal{M}} = A(\bigcup_{q \in G} \Pi(q)),$$

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

$$\Pi_{\mathcal{M}} = E(\bigcup_{q \in G} \Pi(q)),$$

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

■ For a recurrence automaton, we are given that $P = \phi$, and therefore $\Pi_{\mathcal{M}} = R(\bigcup_{q \in R} \Pi(q))$.

■ For a persistence automaton, we are given that $R = \phi$, and therefore $\Pi_{\mathcal{M}} = P(\bigcup_{q \in P} \Pi(q))$.

Consider now the other direction of the proposition. It states that a $\kappa$-property specifiable by automata can be specified by a $\kappa$-automaton. Assume that a $\kappa$-property $\Pi$ is specifiable by automata. Thus, there exists a Streett automaton

$$\mathcal{M} = \langle Q, q_0, T, L\rangle, \quad L = \{(R_i, P_i), \quad i = 1, \ldots, k\}$$

specifying $\Pi$.

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

■ Consider first the case that $\Pi$ is a *safety* property, and hence, satisfies $\Pi = A(Pref(\Pi))$.

We construct an automaton:

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

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

$$\begin{array}{rcl} G & = & \{q_0\} \cup \{q \in Q \mid \begin{array}{l} \delta(q_0, \sigma) = q \text{ for some} \\ \sigma \in Pref(\Pi) \end{array} \}, \\ B & = & Q - G. \end{array}$$

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

$$t'(q, q') = \begin{cases} \mathbf{T} & q \in B,\ q' = q \\ \mathbf{F} & q \in B,\ q' \neq q \\ t(q, q') & q \notin B \end{cases}$$

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

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

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 $\sigma$ cannot be a prefix of a computation in $\Pi$. 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 $\sigma' \cdot \sigma'' \in \Pi$. Consider the mixed computation $\sigma \cdot \sigma'' \in \Sigma^\omega$. Let $r$ be the run of $\langle Q, q_0, T\rangle$ over $\sigma \cdot \sigma''$, and $r'$ the run of $\langle Q, q_0, T\rangle$ over $\sigma' \cdot \sigma''$. 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 $\sigma \cdot \sigma''$ should be accepted. 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 $\delta'$ the transition function based on $T'$. Assume that $\sigma$ 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 $\mathcal{M}'$ 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 $\Pi$ is a safety property, that $\sigma \in \Pi$.

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

■ Consider the case that $\Pi$ is a *guarantee* property.

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

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

Construct the automaton:

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

where $T'$ is given by:

$$t'(q, q') = \begin{cases} \mathbf{T} & q \in G,\ q' = q \\ \mathbf{F} & q \in G, q' \neq q \\ t(q, q') & q \notin G \end{cases}$$

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

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

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

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

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

Without loss of generality, we define the modifications on the first pair $R_1, P_1$. After obtaining a $P_1' = \phi$, 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 $\mathcal{C}$, 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 $t(q_i, q_j) = \mathbf{F}$. 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 $\mathcal{C} \cap R_1 = \phi$, and hence $\mathcal{C} \subseteq P_1$. Define $A_1$ to be the set of automaton states participating in persistent cycles.

Let $\mathcal{M}$ be the automaton accepting $\Pi$ with accepting pairs $(R_i, P_i)$, $i = 1, \ldots, k$, and consider the automaton $\mathcal{M}'$ coinciding with $\mathcal{M}$ in all but the accepting pairs. The list of accepting pairs for $\mathcal{M}'$ is $(R_1', P_1')$, $(R_i, P_i)$, $i = 2, \ldots, k$, where we define:

$$\begin{aligned} R_1' &= R_1 \cup A_1 \\ P_1' &= \phi. \end{aligned}$$

We wish to show that $\mathcal{M}$ and $\mathcal{M}'$ accept precisely the same computations.

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

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

The state $q$ and computation $\sigma'$ have the following property:

For every finite computation $\hat{\sigma}$, such that $\delta(q_0, \hat{\sigma}) = q$, there exists a positive integer $n$ (possibly dependent on $\hat{\sigma}$) such that $\hat{\sigma} \cdot (\sigma')^n$ contains a prefix $\sigma \prec \hat{\sigma} \cdot (\sigma')^n$, $|\sigma| > |\hat{\sigma}|$, which belongs to $\Pi'$.

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

Let now $\sigma_0$ be a computation such that $\delta(q_0, \sigma) = q$, and $\hat{\sigma}$ a computation leading the automaton from $q$ to $q$ along the path $\pi$, which precisely traverses $J$. Consider the following infinite computation:

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

where the $n_j$'s are chosen so that $\sigma''$ has infinitely many prefixes in $\Pi'$. 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 $\Pi'$, longer than $\sigma''_{j-1}$.

It follows, on one hand, that since $\sigma''$ has infinitely many prefixes in $\Pi'$, $\sigma'' \in \Pi$.

On the other hand, the infinity set corresponding to $\sigma''$ is $J \cup J'$ which has an empty intersection with $R_1$ and at least one state $q' \notin P_1$. This contradicts the assumption that $\mathcal{M}$ specifies $\Pi$.

Consequently, there cannot exist a computation $\sigma$ which is accepted by $\mathcal{M}'$ and rejected by $\mathcal{M}$.

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

It only remains to show that such an automaton is equivalent to an automaton with a single $R$ and a single $P = \phi$. This is essentially a closure property that states that the intersection of recurrence automata is equivalent to a single recurrence automaton. The construction is similar in spirit to the 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 $\Pi$ 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

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

specifying $\overline{\Pi}$. The following persistence automaton obviously specifies $\Pi$

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

■ The case of reactivity properties specifiable by automata is handled as follows.

Let $\Pi$ be a reactivity property specifiable by the automaton

$$\mathcal{M} = \langle Q, q_0, T, \{(R_i, P_i), \quad i = 1, \ldots, k\} \rangle.$$

Clearly the role of the list of pairs $(R_i, P_i)$, $i = 1, \ldots, k$, is to define the subsets $J \subseteq Q$ such that every computation $\sigma$ with $\inf(r(\sigma)) = J$ is accepted. Let $F$ denote the family of these sets. Obviously, $J \in F \longleftrightarrow (R_i \cap J \neq \phi$ or $J \subseteq P_i)$ for each $i = 1, \ldots, k$.

A characterization property that can be derived from Wagner [Wag79] (see also [Kam85]) is the following:

If $\mathcal{M}$ specifies a reactivity 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 $X$, $A_i \subseteq X \longrightarrow X \in F$ and for each $B_j$, and an arbitrary accessible cycle $X$, $X \subseteq B_j \longrightarrow X \in F$.

This leads to the construction of the following automaton:

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

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

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 $\mathcal{M}$ 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 in $q'$ 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 the $A_i$'s 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 $\mathcal{M}$ stays completely within one of the sets $B_1, ..., B_n$ 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 $f_P$ is 0.

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

- $T'$ is defined as follows:

$$\begin{aligned} t'(\langle q, q_1, \ldots, q_m, f_R, j, f_p \rangle,\ & \langle \tilde{q}, \tilde{q}_1, \ldots, \tilde{q}_m, \tilde{f}_R, \tilde{j}, \tilde{f}_P \rangle) = \\ & t(q, \tilde{q}) \\ \wedge & \bigwedge_{i=1}^{m} \{ \ ((\tilde{q} = q_i) \wedge (\tilde{q}_i = next(q_i, A_i))) \vee \\ & \qquad ((\tilde{q} \neq q_i) \wedge (\tilde{q}_i = q_i)) \} \\ \wedge & \ ((\tilde{f}_R = 1) \equiv \bigvee_{i=1}^{m} (\tilde{q} = q_i)) \\ \wedge & \ ( \ ((\tilde{q} \in B_j) \wedge (\tilde{j} = j)) \vee \\ & \qquad ((\tilde{q} \notin B_j) \wedge (\tilde{j} = [j \ mod\, m] + 1))) \\ \wedge & \ ((\tilde{f}_P = 1) \equiv (\tilde{q} \in B_j)). \end{aligned}$$

- 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 state of $A_i$, and then we modify $q_i$ to the next $A_i$-state in sequence, or there is no match and then $q_i$ remains the same.
- The third clause states that $f_R$ is set to one iff $\tilde{q}$ matches one of the anticipated 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 by one in a cyclic manner.
- The last clause states that $f_P$ is set to 1 whenever $\tilde{q}$ is in $B_j$.

- The acceptance sets are defined by

$$\begin{aligned} R' &= \{\langle q, q_1, \ldots, q_m, f_R, j, f_P\rangle \in Q' \mid f_R = 1\} \\ P' &= \{\langle q, q_1, \ldots, q_m, f_R, j, f_P\rangle \in Q' \mid f_P = 1\}. \end{aligned}$$

Let $\sigma$ be a computation and $r'$ the corresponding run of $\mathcal{M}'$ over $\sigma$. If $r'$ visits $R'$ infinitely many times, this implies that $r$, the run of $\mathcal{M}$ over $\sigma$, visits infinitely many times *all* the states of some $A_i$. This shows that $\inf(r) \supseteq A_i$ and hence $\sigma$ is accepted by $\mathcal{M}$ as well as by $\mathcal{M}'$.

If $r'$ stays contained in $P'$ from a certain point on, it 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 $\sigma$ is accepted by $\mathcal{M}$ as well as by $\mathcal{M}'$.

As similar argument shows that all computations accepted by $\mathcal{M}$ are also accepted by $\mathcal{M}'$. ▪

## 5.1 Deciding the Type of a Property

In this section we consider the following problem:

**Problem 5.1** *Given a Streett automaton $\mathcal{M}$, decide whether the property specified by this automaton is a $\kappa$-property, where $\kappa \in \{$safety, guarantee, obligation, recurrence, persistence, reactivity$\}$*

The following proposition gives an answer to this general question:

**Proposition 5.2** *It is decidable whether a given Streett automaton specifies a property of type $\kappa$, where $\kappa \in \{$safety, guarantee, recurrence, persistence, reactivity$\}$.*

Again, for the first types, the answer has been given by Landweber in [Lan69]. For the case of reactivity, as well as the complete hierarchy below, it is provided by Wagner in [Wag79].

In the context of specification, this question was tackled in [AS87], where a decision procedure is given for safety and 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 $A \subset Q$ is defined to be *closed* if for every $q, q' \in Q$

$$q \in A \ \wedge \ t(q, q') \neq \mathbf{F} \ \longrightarrow \ q' \in A$$

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

For a given Streett 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 $\mathcal{M}$ specifies a safety property iff $\hat{B} \cap G = \phi$.

- Checking for a *guarantee* property.
  $\mathcal{M}$ specifies a guarantee property iff $\hat{G} \cap B = \phi$.

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

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

The following are direct consequences of the characterizations in [Wag79]:

- 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 a *reactivity* property.
  $\mathcal{M}$ specifies a reactivity property iff there do not exist three accessible cycles

  $$B \subseteq J \subseteq A$$

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

As a matter of fact, the methods of [Wag79] identify the exact location of an automaton specifiable property in the reactivity hierarchy, i.e., the minimal $k$ such that the property can be specified by a Streett 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 $i = 1, \ldots, n$.

## 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 5.3** *A property that is specifiable by a $\kappa$-formula is specifiable by an $\kappa$-automaton, for $\kappa$ ranging over the different types.*

This is based on the following construction, studied in [LPZ85] and [Zuc86].

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 $F_1, \ldots, F_k \subseteq Q$. The automaton $\mathcal{M}$ has the property that for each $i = 1, \ldots, k$, each infinite computation $\sigma \in \Sigma^\omega$, and each position $j \geq 0$,

$$\delta(q_0, \sigma[0 \ldots j]) \in F_i \quad \textit{iff} \quad (\sigma, j) \models p_i\ .$$

Thus, the automaton $\mathcal{M}$ identifies, while reading $\sigma$ up to position $j$, which $p_i$'s hold at that position.

Using this basic construction, it is straightforward to build a $\kappa$-automaton corresponding to a $\kappa$-formula.

For example, for the reactivity formula $\Box\Diamond p_1 \vee \Diamond\Box p_2$, let the automaton mentioned above be $\langle Q, q_o, T\rangle$ with the designated sets $F_1$ and $F_2$. Then the corresponding reactivity 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 [MP71]) can be translated into temporal logic. A (Street) automaton is defined by be counter-free if there exists no finite computation $\sigma$ and a 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 $\sigma$ would have enabled the automaton to count occurrences of $\sigma$ modulo $n$.

It has been shown in [Zuc86] 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 $\kappa$-automata to $\kappa$-formulae.

**Proposition 5.4** *A property that is specifiable by a counter-free $\kappa$-automaton is specifiable by a $\kappa$-formula.*

The translation is essentially the one studied in [Zuc86], but showing that the structure required in a $\kappa$-automaton corresponds to the structure required in a $\kappa$-formula.

It is based on the construction of a past formula $\varphi_q$ for each $q \in Q - \{q_0\}$ of a given counter-free automaton table $\langle Q, q_0, T\rangle$. The formula $\varphi_q$ characterizes all the finite computations leading from $q_0$ to $q$, i.e., for each infinite computation $\sigma \in \Sigma^\omega$ and position $j \geq 0$,

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

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

$$\Box\Diamond\left(\bigvee_{q\in R} \varphi_q\right) \quad \vee \quad \Diamond\Box\left(\bigvee_{q\in P} \varphi_q\right).$$

The above two ways translation, subject to counter-freedom, provides a standard reduction of results about automata into the corresponding results about temporal logic. We can use this reduction to prove the other direction of the claim relating the temporal classes to their semantic specification.
We illustrate this method on the following part of the proposition.

> A reactivity property $\Pi$ that is specifiable by temporal logic, is specifiable by a reactivity formula.

### Proof

Let $\varphi$ be the formula specifying $\Pi$. Using the translation described in proposition 5.3, we construct a counter-free automaton $\mathcal{M}_\varphi$, specifying the reactivity property $\Pi$. Using the construction described in proposition 5.1 for the case of a reactivity property, we construct a reactivity automaton $\tilde{\mathcal{M}}$ that specifies the same property. The construction of $\tilde{\mathcal{M}}$ 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 $\tilde{\mathcal{M}}$. We can now use the translation from counter-free automata to temporal formulae, described in proposition 5.4, to construct a reactivity formula $\varphi_{\tilde{\mathcal{M}}}$ specifying $\Pi$.

This method was used in [Zuc87] to establish the strict hierarchy for temporal formulae, based on [Kam85].

# References


[AS85] B. Alpern and F.B. Schneider, Defining liveness, *Info. Proc. Lett.* **21**, 1985, pp. 181–185.

[AS87] B. Alpern and F.B. Schneider, Recognizing safety and liveness, *Dist. Comp.* **2**, 1987, pp. 117–126.

[AS89] B. Alpern and F.B. Schneider, Verifying temporal properties without temporal logic, *ACM Trans. Prog. Lang. Sys.* **11**, 1989, pp. 147–167.

[Kam85] M. Kaminski, A classification of $\omega$-regular languages, *Theor. Comp. Sci.* **36**, 1985, pp. 217–229.

[Lam77] L. Lamport, Proving the correctness of multiprocess programs, *IEEE Trans. Software Engin.* **3**, 1977, pp. 125–143.

[Lam83] L. Lamport, What good is temporal logic, *Proc. IFIP 9th World Congress* (R.E.A. Mason, ed.), North-Holland, 1983, pp. 657–668.

[Lan69] L.H. Landweber, Decision problems for $\omega$-automata, *Math. Sys. Theory* **4**, 1969, pp. 376–384.

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

[Man74] Z. Manna, *Mathematical Theory of Computation*, McGraw-Hill, 1974.

[MP71] R. McNaughton and S. Papert, *Counter Free Automata*, MIT Press, 1971.

[MP83] Z. Manna and A. Pnueli, How to cook a temporal proof system for your pet language, *Proc. 10th ACM Symp. Princ. of Prog. Lang.*, 1983, pp. 141–154.

[MP84] Z. Manna and A. Pnueli, Adequate proof principles for invariance and liveness properties of concurrent programs, *Sci. Comp. Prog.* **32**, 1984, pp. 257–289.

[MP87] Z. Manna and A. Pnueli, Specification and verification of concurrent programs by $\forall$-automata, *Proc. 14th ACM Symp. Princ. of Prog. Lang.*, 1987, pp. 1–12.

[MP89a] Z. Manna and A. Pnueli, The anchored version of the temporal framework, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency* (J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, eds.), Lec. Notes in Comp. Sci. 354, Springer, 1989, pp. 201–284.

[MP89b] Z. Manna and A. Pnueli, Completing the temporal picture, *Proc. 16th Int. Colloq. Aut. Lang. Prog.*, Lec. Notes in Comp. Sci. 372, Springer, 1989, pp. 534–558.

[OL82] S. Owicki and L. Lamport, Proving liveness properties of concurrent programs, *ACM Trans. Prog. Lang. Sys.* **4**, 1982, pp. 455–495.

[Pnu77] A. Pnueli, The temporal logic of programs, *Proc. 18th IEEE Symp. Found. of Comp. Sci.*, 1977, pp. 46–57.

[Rab72] M.O. Rabin, *Automata on Infinite Objects and Churc's Problem*, Volume 13 of *Regional Conference Series in Mathematics*, Amer. Math. Soc., 1972.

[Sis85] A.P. Sistla, On caracterization of safety and liveness properties in temporal logic, *Proc. 4th ACM Symp. Princ. of Dist. Comp.*, 1985, pp. 39–48.

[Str82] R.S. Streett, Propositional dynamic logic of looping and converse is elementarily decidable, *Inf. and Cont.* **54**, 1982, pp. 121–141.

[Wag79] K. Wagner, On $\omega$-regular sets, *Inf. and Cont.* **43**, 1979, pp. 123–177.

[Wol83] P. Wolper, Temporal logic can be more expressive, *Inf. and Cont.* **56**, 1983, pp. 72–99.

[Zuc86] L. Zuck, *Past Temporal Logic*, Ph.D. thesis, Weizmann Institute, 1986.

[Zuc87] L. Zuck, Manuscript, 1987.