# The Anchored Version of the Temporal Framework *

**Zohar Manna**
Department of Computer Science
Stanford University, Stanford, CA 94305
&
Department of Applied Mathematics
Weizmann Institute of Science, Rehovot

**Amir Pnueli**
Department of Applied Mathematics
Weizmann Institute of Science, Rehovot

**ABSTRACT.** In this survey paper we present some of the recent developments in the temporal formal system for the specification, verification and development of reactive programs. While the general methodology remains very much the one presented in some earlier works on the subject, such as [MP83c, MP83a, Pnu86], there have been several technical improvements and gained insights in understanding the computational model, the logic itself, the proof system and its presentation, and connections with alternative formalisms, such as finite automata. In this paper we explicate some of these improvements and extensions.

The main difference between this and preceding versions is that here we consider a notion of validity for temporal formulae, which is *anchored* at the initial state of the computation. The paper discusses some of the consequences of this decision.

**Key words:** Temporal Logic, Reactive Systems, Concurrent Programs, Specification, Verification, Proof System, Classification of Prtoperties, Safety, Liveness, Recurrence, Responsiveness, Progress, Fairness, Fair Transition System.

## Contents



*This research was supported in part by the National Science Foundation under grants DCR-8413230 and CCR-8812595, by the Defense Advanced Research Projects Agency under contract N00039-84-C-0211, and by the United States Air Force Office of Scientific Research under contracts AFOSR 87-0149 and 88-0281.

# 1 Introduction

Over the last several years we have witnessed the evolution of a formal methodology for the specification, verification, and development of reactive programs (and more general reactive systems), based on temporal logic. This methodology consists of several elements which we list below.

- *A computational model*, which provides a general uniform representation for the various programming languages and diverse suggested mechanisms for synchronization and communication between the concurrent processes comprising a reactive system. The generic computational model used in the temporal methodology presented in this paper is that of *Fair Transition System*. The computation model assigns a semantics to each reactive system. This semantics associates with each program a *behavior*, which is a single or a set of computation structures which represent its possible executions. In the case considered here, the semantics of a program is the set of its computation, where each computation is a sequence of states that can be generated in a single execution of the program.

- *A specification language*, which is essentially temporal logic augmented by some program specific predicates and functions, referring to the additional programming constructs needed to fully describe a state in the computation of a reactive program. In order to relate a specification presented by a formula in the logic to the program it is supposed to specify, it is necessary that the computational structures defined to be the semantics of a program can serve as *models* (in the logical sense) for the formula, which means that we can evaluate the formula on each of these structures and find whether it holds (is true) on the structure. Then, we say that the program *satisfies* (or *implements*) the specification given by a formula $\varphi$ if $\varphi$ holds over each of the computation structures. For example, in the linear semantics we consider here, the specification language is *linear* temporal logic, whose models are arbitrary sequences of states. Since the semantics of a program is a set of computations, which are also sequences of states, the specification $\varphi$ is valid over the program $P$ if it holds over all the computations of $P$.

- *Classification of Properties.* While not absolutely necessary, it is often very convenient and instructive to classify the properties that can be expressed in the specification language into several classes which share some common characteristics. After all, we should never forget that, even if our fondest dreams become true, and every constructed program is rigorously verified, the only guarantee this gives us is that the program precisely and unambiguously implements the specification, namely the program is correct relative to the specification. However, how do we know that the *specification* is correct, relative to the informal requirements or intuition about the intended behavior of the system? Formal tools can only help us in comparing two formal presentations of a system, but cannot help us in reconciling a formal description with an informal one. Usually we should be concerned with two aspects of the specification, which are its *consistency* and *completeness*. Consistency of a specification is something that can yield to formal tools, since we are comparing some parts of the specification with some other parts. Completeness, on the other hand, must be judged against the informal requirements. By recognizing some standard classes of properties, which usually occur in systems of a certain kind, we are alerted to the possibility of incompleteness by noticing that a specification of such system does not contain any properties of one of these classes. For example, it is now a folk axiom that no specification of a system can be complete without containing some *safety* properties and some *liveness* properties.

Indeed, one of the most important classifications of properties of reactive systems, is their partition into safety and liveness properties. In this paper, we present a more detailed classification of properties which distinguishes many more subclasses than those two, but which serves the same purpose of structuring the specification and its proof methods into several frequently occurring patterns.

- *A proof system* which enables us to formally prove that a given program satisfies a given specification. Such a proof system can be partitioned into three parts (see [MP83c, MP83a]):
  - ⋆ *A General Part.* This part provides axioms and rules to establish the validity of formulae which are generally valid, i.e., over any model, not necessarily associated with a given program. For example, for linear temporal logic this part is used to establish the validity of formulae that hold over any sequence of states. Thus, this part provides us with the tools to conduct program and domain independent reasoning over temporal models.
  - ⋆ *A Domain Part.* This part provides axioms and rules to reason about the data domain which both the program and the specification refer to. Thus, for a program over the integers, we usually employ a specification which refers to the predicates and operations specific to the integers, such as $>, +$, etc. Similar theories should be provided for other data domains such as strings, lists, etc. While the general part enables us to establish validity of uninterpreted temporal formulae, the combination of the general part and the domain part extends this ability to deal also with interpreted temporal formulae. Still, the validity considered is over arbitrary interpreted sequences of states. An important function of the domain part is that it provides us with multiple induction principles, corresponding to the various well founded domains we may study.
  - ⋆ *A Program Part.* This is the part that restricts the set of considered models to those that correspond to the behavior of the specific program we wish to analyze. By having deduction rules which are specific to a particular program, we can establish temporal properties which are not valid over all possible models, but just over the models associated with the behavior of the program. Together with the more general tools presented in the previous parts, the combined system gives us sufficient power to prove most of the properties that can be specified in temporal logic. This statement is made precise by several completeness results for the proof system.

- *Decision Procedures* and tools for *Automatic Verification.* While the main tool for verifying that a program satisfies its specification is still the deductive proof system discussed above, it depends very much on the skill and ingenuity of the person performing the verification. In some cases, the verification problem is decidable, and then it is possible to perform some verification tasks fully automatically, without any need for human intervention. The most prevalent of these cases is the case of *finite state systems*, in which all the data domains over which the program variables vary are finite. In this case the specification can always be represented by a propositional temporal formula, and the verification problem becomes decidable. Differently from sequential programs, there are many finite state reactive programs whose verification problem is far from trivial. Notably, many synchronization problems can, with some degree of abstraction, be faithfully represented as finite state programs. There has been quite an intense activity over the last several years, concentrating on the construction and efficient implementation of algorithms for automatic verification of finite state programs. See for example [CES86, LP84, QS83], etc.

- *Development* Paradigms. While the ability to verify an existing program relative to its specification is a very important element of a methodology for the systematic development of a program from its specification, it cannot serve as the sole basis for such a methodology. A proper methodology cannot wait until a full implementation is constructed and then attempt to verify it. The more constructive approach is to design a set of development strategies, which gradually transform the specification into a correct implementation. Several such development schemes have been successfully suggested for sequential programs by works such as [Dij76, Gri81], etc., and recently for reactive programs by [CM88]. To carry such a program for the temporal methodology requires a better understanding of the two major transformations that can be considered for reactive systems, which are *decomposition* and *refinement*. By refinement we mean the replacement of relatively moderate size actions, that at a certain level of abstraction can be considered atomic, by a more detailed implementation that necessarily makes them non atomic on the next level. Such a replacement is usually orthogonal to the process structure of the system and may affect uniformly many of the processes at the same time, but does not alter the process structure itself. Some general approaches to this important problem are discussed in [AL88, Bac80] as well as many other places. Decomposition, on the other hand, usually involves breaking some process into composition of smaller processes. To properly handle decomposition, we need a *compositional* proof system, in which the temporal properties of a composite process can be deduced from the properties of its components. We refer the reader to [dRZ87] for discussion of some of the requisites from such a system, and to a sequence of works [BKP84, NGO85, WGS87, Zwi88], which attempt to construct such a compositional proof system.

As can be seen from the short account of the main elements, of which the temporal methodology consists, some of them are in a more advanced stage of development, and are better understood than the others. However, there seems to be an accumulated evidence indicating that this methodology, or a one similar to it, can provide a feasible formal approach to the systematic development of reactive systems. This does not necessarily implies that we expect all parts of a big real time system to be rigorously developed and formally verified. According to a more modest and realistic view, the methodology or various parts of it can be used to develop small but critical parts of the system, most often subject to a certain degree of abstraction and simplification.

While there may be a consensus about this general principle, at least among the afictionados of the formal approach to programming, opinions vary greatly as to the precise details and parameters of the different constituents of the methodology. We will consider below the main parameters involved, and then indicate the choice selected by us and maintained for the rest of the paper.

## 1.1 Some Parameters of the Methodology

### Type of Semantics

One of the important parameters of the methodology is the type of semantics which is assigned to programs. Some of the possible choices are, respectively:

- *Linear Semantics*. Under this choice, the semantics of a program $P$ is the set of all the computations of $P$, where each computation is a (possibly infinite) sequence of states generated by performing the basic actions of the program (transitions) one at a time. Concurrent activity of two parallel processes in the program is represented by the *interleaving* of their atomic actions. To compensate for this much simpler, but perhaps less natural, representation of concurrency by interleaving, we add the notion of *fairness* to the computational model. One of the main functions of the fairness requirements is to exclude the interleaving between two processes in which, beyond a certain point all the actions taken are taken from only one of the processes, while the other has some enabled actions which never get executed. One of the simplifications taken in the linear approach is that we do not retain the information of where did the program take a choice between several possibilities. This can be illustrated by saying that the linear approach considers the program $a(b+c)$ to be (semantically) equivalent to the program $ab + ac$, even though the first program decides between taking $b$ or $c$ only after performing $a$, while the second program decides a priori whether it is going to perform $ab$ or $ac$. Another obvious simplification inherent in the linear approach is its inability to distinguish between the non-determinism caused by a non-deterministic choice existing in the program for one of the processes, and the one introduced by the interleaving representation of concurrency, which in the case of two independent actions $a$ and $b$ being enabled in two processes, can choose to perform $a$ before $b$, or vice versa. This can be illustrated by claiming that the linear approach also considers the programs $ab + ba$ and $a\|b$ to be semantically equivalent, even though, in the first program there is a non-deterministic choice in the program for the single process, while in the second program each of the processes is deterministic, and it is only our attempt to describe their possible behavior by linear sequences which causes the possibility of two such sequences, namely $a;b$ and $b;a$.

- *Branching Semantics.* Under this choice, the semantics of a program $P$ is a single (possibly infinite) tree of states, where each node representing a state $s$ in the computation, has as direct descendants all the states that can be derived from $s$ by the execution of one of the atomic transitions which can be performed from $s$. This semantics certainly distinguishes between the program $a(b+c)$ and the program $ab+ac$, which differ in the point at which the choice between the two possibilities is made. On the other hand, this approach still represents concurrency by interleaving, and considers the programs $a\|b$ and $ab + ba$ equivalent.

- *Partial Order Semantics.* Under this view, the semantics of a program is a (possibly infinite) structure of states (or events), on which we recognize two basic relations. One is a partial order which represents the *precedence* ordering between events, and constrains certain events to occur only *following* some other events. Typically, this will be the case of two actions $a$ and $b$, where $b$ follows $a$ in a sequential program of one of the processes (such as $acb$). The other relation is that of *conflict.* Two events are considered to be conflicting if they can never participate in the same execution. A typical case of conflict is associated with choice. Thus, in the program $ab + cd$, the actions $a$ and $c$ are in conflict. If we execute the action $a$, we lose the possibility of executing the action $c$. The conflict relation is extended by the precedence relation. Thus, if $a$ is in conflict with $c$, and $a$ and $c$, respectively precede $b$ and $d$, it follows that each of $\{a,b\}$ is in conflict with each of $\{c,d\}$. If two events are neither related by the precedence relation, nor are in conflict, we say that they are *independent*, which can be interpreted as being concurrent, i.e., can be executed in parallel. An execution is any maximal substructure which does not contain two conflicting actions. Note that in describing a computation we refuse to commit ourselves to the ordering between actions which are independent. This is the reason that the partial order semantics is the only one from those we consider, which identify concurrency as a unique phenomena which is not translatable to any interleaved representation. To illustrate these concepts, we consider the trivial programs we have mentioned above. For the program $a\|b$, we have two actions, $a$ and $b$, which are identified as independent. For the program $ab + ba$, we have to consider *four* actions $a', b', a^*, b^*$, where $a' \prec b'$ and $b^* \prec a^*$, and each of $\{a', b'\}$ is in conflict with each of $\{b^*, a^*\}$. Clearly, these two programs correspond to completely different structures. Similarly $a(b+c)$ differs from $ab+ac$ because, for example, the structure corresponding to $ab+ac$ contains two actions $a, a^*$, which are in conflict. Thus, it is obvious that the partial order semantics is sensitive both to the location of choice and to concurrency as different from interleaving. We refer the reader to some of the standard references to Petri Nets, such as [Rei85] for a fuller description of this approach.

This variety of semantics naturally leads to a variety of corresponding temporal logics. Consequently, we observe the following proposed logics to deal with each of the semantic views:

- Linear Temporal Logic. This is the logic considered in this paper, and the one studied previously in [GPSS80, MP83c, MP83a]. Tailored for a semantics consisting of a set of linear executions, formulae in this language define predicates over sequences.

- Branching Temporal Logic. Different variants of branching temporal logics have been considered, starting with the simpler logic presented in [BMP83], and culminating in the most expressive $CTL^*$ of [EH86]. All these logics are interpreted on a branching structure that can be viewed as a tree. For a semantics of a program represented by a single computation tree, a formula in the language defines a predicate over such trees.

- Temporal Logic over Partial Orders. Here the situation is much more fluid. Only very recently have there been suggestions for formulating such a logic. Some of them are presented in this volume.

We can summarize the four different combinations of the two distinguishing criteria in the table below:

| | Does not Distinguish Concurrency | Distinguishes Concurrency |
|---|---|---|
| Does Not Distinguishes Branching Points | Linear Semantics<br><br>Linear Temporal Logic | Sets of Partial Order Executions<br>Interleaving Sets Logic [KP87] |
| Distinguishes Branching Points | Computation Trees<br><br>Branching Time Logics | Full Event Structures<br><br>Logics? |

## The Expressive Power of the Specification Language

Even if we settle on the linear type of semantics, there are still many choices of the exact syntactic features and the expressive power of the chosen specification language. Some of the issues and alternatives that can be considered are listed below.

- One can choose (as a starting point) the future fragment of the temporal logic, including the *Until* and *Next* operator, as suggested in [GPSS80], and further developed in [MP83c] to include quantification over *static* (called there *global*) variables.

- Various arguments have been advanced against the use of the *Next* operator, claiming that it can express distinctions (such as whether an identical state repeats twice or three times in succession), which a properly abstract semantics should not make. This has been claimed in [Lam83b] which provides strong evidence that all the properties one wishes to express for asynchronous systems do not require this operator.

- Another important observation was made in [Wol83] pointing that there are simple properties, usually referring to counting events, which are not expressible in the regular version of temporal logic. This makes temporal logic less expressive than the formalism of $\omega-$automata which is another attractive formalism for expressing properties of sequences. Wolper suggested in [Wol83] to remedy this problem by the introduction of special *grammar operators*. Meanwhile, there have been additional suggestions for extending temporal logic to attain the full expressive power of $\omega-$automata. Some of these suggestions include the use of quantifiers over *dynamic* variables (also referred to as *local*), and the use of fixpoint operators. Both of these extensions leave the propositional version of the logic decidable, even though undisciplined use of quantifiers may significantly boost up the complexity of the decision procedure. Quantification over dynamic variables turned out to be a very useful device for abstraction, and has been shown to be necessary for attaining compositional completeness.

- An interesting alternative to temporal logic is specification by $\omega-$automata. The utility of automata for automatic verification of finite state programs has been long realized (see [VW86]). However, a relatively recent development, suggested first in [AS89], generalized the use of automata to specify also infinite state programs (see also [MP87]). We will expand on this alternative later in the paper.

- Somewhat related to the automata approach, is the suggestion to use *fair transitions systems* not only, as we do, as a computational model, but also as a specification language. This suggestion has a strong appeal of uniformity, in that the specification and implementation languages coincide, and we can immediately conceive of a multi-level process of refinement from a most abstract transition system to a concrete one, at the end, which defines the implementation. This approach to specification and associated development has been suggested by many researchers, including [Lam83a, Jon87, BK83]. In some sense we can view [CM88] as one of the most complete accounts of specification and refinement by fair transition systems.

## 1.2 The Anchored Version of Temporal Logic

The original version of the temporal language presented in [GPSS80], and which serves as the basis for the temporal methodology described in [MP83c, MP83a, Pnu86], is based on two major design decisions:

- Only the future fragment is considered. In that the approach deviates from Kamp's version of temporal logic ([Kam68]), which serves as the major inspiration for [GPSS80]. Kamp considers a language with both future and past operators. The decision to adopt only the future fragment was based on the fact, established in [GPSS80], that this fragment by itself is expressively complete, meaning that it can express any property expressible by temporal logic (and the first order theory of linear order, which serves as the standard yardstick for temporal expressibility). It also appeared to provide natural expressions for the main classes of program properties one may wish to formalize. For example:

  - The formula $\Box\, p$ expresses the invariance of the assertion $p$ over all executions of the program.
  - The formula $p \rightarrow \Diamond\, q$ expresses an eventuality property that guarantees the single occurrence of the event $q$, such as termination of the program that is expected to occur only once.

- The formula $\Box(p \rightarrow \Diamond q)$ expresses a repeated eventuality property (also called *response* properties), which guarantees that *every* occurrence of $p$ must be followed by an occurrence of $q$.
- The formula $\Box(p \rightarrow (\neg r)\mathcal{W}q)$, using the *Unless* operator $\mathcal{W}$ (a weak version of the *Until* operator $\mathcal{U}$), expresses a (continuous) precedence property, ensuring that from any occurrence of $p$, the first following occurrence of $r$ (if any) must be preceded by an occurrence of $q$ (see [MP83b] for applications of such formulae).
- The formula $\Box \Diamond p \rightarrow \Box \Diamond q$ expresses a strong fairness property, ensuring that if there are infinitely many $p'$s then there are also infinitely many $q'$s. This property can represent a strong fairness requirement, which states that if a certain transition is infinitely often enabled, then it is also infinitely often taken. It can also represent a situation of response to persistence, in which the response $q$ is guaranteed only if the request $p$ is repeated sufficiently many times (infinitely many in the limit).

- A second basic feature of the [GPSS80] temporal logic is that satisfiability and validity are evaluated at *all* positions in the computation. Thus, a valid formula is expected to hold at all positions of a computation, and a formula is considered to be satisfiable if it holds at *some* position in some computation. This view treats all states in the computation uniformly and does not assign any special significance to the initial state. The best representation of this view is by the *Generalization* rule of the proof system of [GPSS80], given by:

$$p \vdash \Box p$$

  This rule claims that if the formula $p$ is valid (anywhere) then it is valid at all states of the computation, and in fact *always* $p$ is also valid. We refer to this feature of the logic as the *floating* interpretation of formulae.

The first indications that something can be gained by relaxing one or both of these design decision came when it was realized that some properties, in particular *precedence* properties, have a more natural expression, when using the past operators. For example, consider the property of *the absence of unsolicited response*, which states that the response $q$ can only happen if it was preceded by the request $p$. Assuming, for simplicity, that both $p$ and $q$ can occur at most once, this is expressed in the [GPSS80] logic as

$$init \rightarrow (\neg q)\mathcal{W}p$$

Note that because of the floating interpretation of the logic, we cannot use the simpler $(\neg q)\mathcal{W}p$ expression. This is because, even in a good computation, i.e., one in which $q$ is preceded by a $p$, there are some positions, namely all those which come after $p$ and before $q$, at which it is not true that the first following occurrence of $q$ must be preceded by an occurrence of $p$. Thus, to state this property we must anchor the observation point, at which the formula is evaluated, to the beginning of the computation. This is done by the special state predicate *init*, which characterizes the initial point in the computation. However, if we are allowed to use past operators in our logic, we may express the same property by the more intuitive formula ([KdR83, KVdR83])

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

which uses the *Once in the past* operator $\diamondminus$, to claim that every $q$ is preceded by a $p$. This form, which states that there is no effect without a cause, also has the aesthetically pleasing symmetry with the formula $\Box(p \rightarrow \Diamond q)$, stating that there is no cause without effect.

Indeed, when carrying out a more thorough investigation of what is involved with adding the past fragment to temporal logic ([LPZ85]), it has been found that, while it does not (mathematically) increase the expressive power of the language, it does make the following important contributions to the utility of the language:

- Many properties, such as the one illustrated above, have much more natural expression when using the past.
- The past operators enable a syntactic characterization of an important classification of the properties expressible by temporal logic into several classes. This classification is very helpful for structuring the specification of a complex system, and for deriving a correlated system of proof principles specifically tailored for each of the classes. These are most useful for guiding the user to use for each property the proof strategy most appropriate for its class.
- The past forms (in particular the normal form on which the mentioned classification is based) are very useful for the establishment of the close connections between temporal logic and transition based formalisms such as $\omega-$automata.
- The relative cost of this expansion of the logic, has been found to be quite modest. The decision procedure for the propositional segment has the same complexity as that of the pure future fragment. A deductive system, proven complete (for the propositional segment), and a model checking procedure have been derived by a straightforward extension of the same constructs for the future logic.

We intend to illustrate some of these advantages in the following sections.

For awhile, we have experimented with the full logic (consisting of both past and future), still retaining the floating interpretation of formulae. However, it soon became apparent that this second basic assumption should also be removed when expanding the logic. We list below some points of dissatisfaction with the floating interpretation, not all of which are directly connected to the introduction of the past.

- The generalization rule $p \vdash \Box p$ is a continuous source of irritation. Technically, it invalidates the important *deduction* rule, which is an important tool in the predicate calculus, claiming that

  $$p \vdash q \text{ iff } \vdash p \rightarrow q.$$

  To see this it is sufficient to observe that $p \vdash \Box p$ according to the generalization rule itself, yet $p \rightarrow \Box p$ is not a generally valid formula.
- The proof system should be applicable not only to the universal set of all possible sequences, but also to more restricted sets of models, for example to those which correspond to computations of a given program. Thus, in addition to the notions of unrestricted validity and provability, which we may denote by $\models p$ and $\vdash p$, respectively, we are also interested in the same notions relativized to a restricted set of models $\mathcal{C}$, and denoted by $\mathcal{C} \models p$ and $\mathcal{C} \vdash p$, respectively. It is easy to see that while the generalization rule is sound over the universal set of models, it is not valid over an arbitrary set of models $\mathcal{C}$. To retain the generalization rule in such a relativization, it is necessary to require that $\mathcal{C}$ be *suffix closed*, i.e., contain with each sequence $\sigma$, all the suffixes of $\sigma$. This is the reason why, when working with the floating framework, we always define the set of computations, over which the temporal formulae are

evaluated, to consist of all the *initialized* computations of the program, together with all their suffixes. One may easily argue that this not a very natural definition.

For example, let us reconsider the property of no effect without cause, which requires that, every computation containing a $q$ at some position, should contain a $p$ at some preceding position. While we would expect this property to hold over all initialized computations of a good program which is expected to satisfy this property, we certainly cannot expect it to hold also over suffixes of such computations. This is because some suffix may contain the $q$ event, but start only after the $p$ event which existed in the full initialized computation. Thus, the true expression of this property in the floating framework must be

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

again relying on the special *init* predicate to restrict our attention to initialized computations only.

- While extra effort, expressed by the need for the *init* precondition, is required to represent anchored properties, such as the precedence property mentioned above or the eventual occurrence of an event that happens only once, it is too easy to represent really floating properties, i.e., those that do hold at all positions of all computations. For example, we are used to express the fact that $p$ happens infinitely many times by the formula $\Box \Diamond p$. However, according to the floating interpretation it is sufficient to claim (and prove) that $\Diamond p$ is valid over the program, since by the floating interpretation (and the generalization rule) there is always an implicit $\Box$ operator standing in front of every valid formula. This is at times both inconvenient and misleading.

Based on these arguments, we have attempted in some of our later works (e.g., [MP87]) to operate in the anchored framework of temporal logic, using the full logic with both the past and future operators. As long as we do not present the full proof system for the logic, the differences between the floating and the anchored versions are not very apparent. They are mainly felt in the following points:

- Suffix closure is no longer required for the set of computations. We only consider initialized computations.

- A formula $\varphi$ is defined to be *valid* over a set of sequences $\mathcal{C}$, if it holds at position 0 of *every* sequence of $\mathcal{C}$.

  The formula $\varphi$ is defined to be *satisfiable* over $\mathcal{C}$, if it holds at position 0 of *some* sequence of $\mathcal{C}$.

- The default mode for expressing properties is the anchored mode, which does not require the special predicate *init*. For example, consider a sequential program with terminal location $l_t$. We will use $l_t$ also as a state predicate, stating that the control is currently at $l_t$. To express unconditional termination of such a program, we write simply $\Diamond l_t$. To express termination conditional on some data precondition $p$ holding at the beginning of the computation, we write $p \rightarrow \Diamond l_t$. To express the precedence property, claiming that every $q$ is preceded by a $p$, we write $\Box(q \rightarrow \diamondminus p)$. On the other hand, to express floating properties, such as $\Box \Diamond p$, we must always include the preceding $\Box$ operator.

In these limited experiments, we found the anchored version much more systematic and unambiguous. Encouraged by this, we present in this paper most of the relevant proof system for supporting the anchored version.

# 2 Programs and Computations

## 2.1 An Abstract Computational Model

The basic computational model we use to represent programs is that of a *fair transition system* ([MP83a, Pnu86]). According to this model, a program $P$ consists of the following components.

- $V = u_1, ..., u_n$ – A finite set of *state variables*. Some of these variables represent *data* variables, which are explicitly manipulated by the program text. Other variables are *control* variables, which represent, for example, the location of control in each of the processes in a concurrent program. We assume each variable to be associated with a domain, over which it varies.

- $\Sigma$ – A set of *states*. Each state $s \in \Sigma$ is an interpretation of $V$, assigning to each variable $y \in V$ a value over its domain, which we denote by $s[y]$.

- $\mathcal{T}$ – A finite set of *transitions*. Each transition $\tau \in \mathcal{T}$ is associated with an assertion $\rho_\tau(V, V')$, referring to both an unprimed and a primed version of the state variables. The purpose of a transition assertion is to express a relation between a state $s$ and its successor $s'$, and we use the unprimed version to refer to values in $s$, and the primed version to refer to values in $s'$. For example, the assertion
$$x' = x + 1$$
states that the value of $x$ in $s'$ is greater by 1 than its value in $s$.

- $\Theta$ – The *initial condition*. This is an assertion characterizing all the *initial* states, i.e., states at which the computation of the program can start. A state satisfying $\Theta$ is called an initial state.

- $\mathcal{W} = \{W_1, ..., W_r\}$ – A family of *weak fairness* requirements (called *justice* requirements in [MP83a, Pnu86]). Each requirement $W_i = (E_i, T_i) \in \mathcal{W}$ consists of two sets of transitions, $E_i, T_i \subseteq \mathcal{T}$, on which the requirement of joint weak fairness is imposed. Intuitively, this forbids a computation in which, beyond a certain point, transitions in $E_i$ are continuously enabled, but no transition of $T_i$ is taken.

- $\mathcal{F} = \{F_1, ..., F_t\}$ – A family of *strong fairness* requirements (called *fairness* requirements in [MP83a, Pnu86]). Each requirement $F_i = (E_i, T_i) \in \mathcal{F}$ consists of two sets of transitions, $E_i, T_i \subseteq \mathcal{T}$, on which the requirement of joint strong fairness is imposed. Intuitively, this forbids a computation in which, beyond a certain point, transitions in $E_i$ are enabled infinitely many times, but no transition of $T_i$ is taken.

We define the state $s'$ to be a $\tau$-*successor* of the state $s$ if
$$\langle s, s' \rangle \models \rho_\tau(V, V'),$$
where $\langle s, s' \rangle$ is the joint interpretation which interprets $x \in V$ as $s[x]$, and interprets $x'$ as $s'[x]$. Following this definition, we can view the transition $\tau$ as a function $\tau : \Sigma \mapsto 2^\Sigma$, defined by:
$$\tau(s) = \{s' \mid s' \text{ is a } \tau\text{-successor of } s\}$$
We say that the transition $\tau$ is *enabled* on the state $s$, if $\tau(s) \neq \phi$. Otherwise, we say that $\tau$ is *disabled* on $s$. We say that a state $s$ is *terminal* if all the transitions $\tau \in \mathcal{T}$ are disabled on it. The enableness of a transition $\tau$ can be expressed by the formula
$$En(\tau) : (\exists V')\rho_\tau(V, V'),$$

which is true in $s$ iff $s$ has some successor.

For a set of transitions $T \subseteq \mathcal{T}$, we say that $T$ is enabled on $s$ if *some* transition $\tau \in T$ is enabled on $s$, and that $T$ is disabled on $s$ if *all* transitions $\tau \in T$ are disabled on $s$.

Given a program $P$ for which the above components have been specified, we define a *computation* of $P$ to be a finite or infinite sequence of states

$$\sigma : s_0, s_1, s_2, \ldots,$$

satisfying the following requirements:

- **Initiality** The state $s_0$ is inital, i.e., $s_0 \models \Theta$.
- **Consecution** For each $j = 0, 1, \ldots,\ s_{j+1} \in \tau(s_j)$, for some $\tau \in \mathcal{T}$. In this case, we say that the transition $\tau$ is *taken* at position $j$ in $\sigma$. For a set of transitions $T \subseteq \mathcal{T}$, we say that $T$ is taken at position $j$, if some $\tau \in T$ is taken at $j$.
- **Termination** Either $\sigma$ is infinite, or it ends in a state $s_k$ which is terminal.
- **Weak Fairness** For each $(E_i, T_i) \in \mathcal{W}$ it is required that, if $E_i$ is continuously enabled beyond some point in $\sigma$, then $T_i$ must be taken at infinitely many positions in $\sigma$.
- **Strong Fairness** For each $(E_i, T_i) \in \mathcal{F}$ it is required that, if $E_i$ is enabled on infinitely many states of $\sigma$, then $T_i$ must be taken at infinitely many positions in $\sigma$.

Note that, by the definition of a transition $\tau$ being taken at a position, it is possible for more than one transition to be considered as taken at one position. Note also that any finite computation automatically satisfies all the weak and strong fairness requirements. For a finite computation $\sigma$, we denote by $|\sigma|$ the length of the computation, i.e., the number of states it contains. For an infinite computation, we write $|\sigma| = \omega$.

For a program $P$, we denote by $\mathcal{C}(P)$ the set of all computations of $P$.

## 2.2 Concrete Model I: Shared Variables

In general, we prefer in this exposition not to repeat material that has appeared previously, and prefer to refer the reader to [Pnu86] which provides a wide coverage of the temporal framework. However, we do present topics that have been refined (and hopefully improved) over the version appearing in [Pnu86]. One of them is the computational model, both abstract and concrete, in which we have refined the notions of fairness requirements. This is why we present several concrete models that illustrate, in particular, the meaning of the fairness requirements.

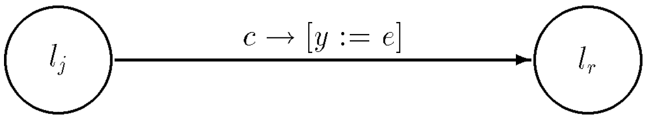
In the shared variables model, a program has the form:

$$P : \{\varphi\}[P_1 \| \ldots \| P_m]$$

Each program is associated with a set of shared variables $Y = y_1, \ldots, y_n$.

the formula $\varphi$, called the *data precondition* of $P$, is an assertion restricting the initial values of the variables in $Y$.

Each $P_i$, called a process, is a sequential program that may access and modify each of the shared variables in $Y$. We will often represent each $P_i$ as a *transition diagram*. A transition diagram for the process $P_i$ is a directed graph with nodes $\mathcal{L}^i = \{l_0^i, \ldots, l_t^i\}$, called *locations*. the location $l_0^i$ is considered to be the entry location, and optionally $l_t^i$ is a stopping location. Each edge in the graph is labeled by a guarded instruction of the form:




where $c$ is a boolean expression (condition), $y \in Y$ is a variable, and $e$ is an expression. We also allow multiple assignment $\overline{y} := \overline{e}$, in which a list of expressions $\overline{e} = e_1, ..., e_k$ is evaluated and simultaneously assigned to a corresponding list of variables $\overline{y} = y_1, ..., y_k$. The intended meaning of such an instruction is that when $c$ holds, execution may proceed from $l_j$ to $l_r$ while assigning the current value of $e$ to $y$.

Alternately, we may choose to represent processes by a text program, assuming the correspondence between the two representations to be obvious.

To assign to a program such as this the semantics of a fair transition system, we identify as follows:

- $V = Y \cup \{\pi_1, ...\pi_m\}$. As state variables we take all the data variables $Y$, to which we add a set of control variables $\pi_1, ..., \pi_m$. Each variable $\pi_i$ for $i = 1, ..., m$ ranges over $\mathcal{L}^i$ and indicates the current location of control for the process $P_i$.

- As states we take all interpretations of $V$ over the appropriate domain.

- As transitions we take all the edges in the diagrams for the processes. Consider a typical transition $\tau$ associated with an edge which connects location $l_j$ to location $l_r$ in process $P_i$, and labeled by the instruction $c \rightarrow [y := e]$. For such a transition we define ${}^\bullet\tau$, called the *source* of $\tau$ to be $l_j$, and $\tau^\bullet$, called the *target* of $\tau$, to be $l_r$. For such a transition, we define the transition assertion $\rho_\tau$ by:

$$\rho_\tau \;:\; (\pi_i = l_j) \wedge c \wedge (\pi_i' = l_r) \wedge (y' = e) \wedge \bigwedge_{u \in V - \{\pi_i, y\}} (u' = u)$$

  Thus, this assertion states that, in order for the assertion to be enabled, it is necessary that $P_i$ be at $l_j$, and that $c$ holds, and when taken, the new value of $\pi_i$ is $l_r$, the new value of $y$ is the current value of $e$, and all other variables retain their values.

  Two transitions, $\tau$ and $\tau'$, are defined to be *in conflict*, if they share the same source, i.e., ${}^\bullet\tau ={}^\bullet \tau'$. For a transition $\tau$, we define the *competition set* of $\tau$, denoted by $Comp(\tau)$ to be the set consisting of $\tau$ itself and all the transitions that are in conflict with $\tau$.

- The initial condition is defined by:

$$\Theta \;:\; \varphi \wedge (\pi_1 = l_0^1) \wedge ... \wedge (\pi_m = l_0^m)$$

  Thus, it is required that initially, the data precondition $\varphi$ holds, and all processes reside in their entry locations.

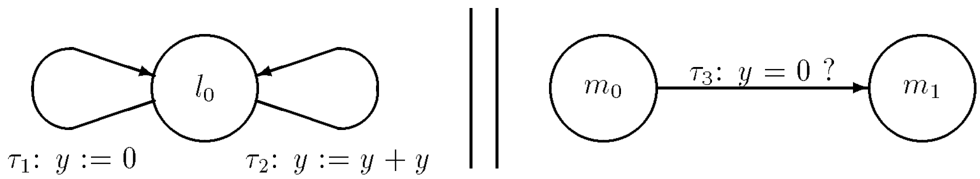
- The weak fairness family includes the requirement $(\{\tau\}, Comp(\tau))$ for each $\tau \in \mathcal{T}$. Thus, we require that if a certain transition is continuously enabled beyond a certain point, then, either this transition, or one competing with it must be taken infinitely many times.

- The strong fairness family is empty at this point, implying that no strong fairness requirements are made yet.

We would like to explain why the weak fairness requirements assume the particular form presented above, rather than the simpler form $(\{\tau\}, \{\tau\})$, requiring that if $\tau$ is continuously enabled then $\tau$ itself should be taken infinitely many times. The reason is that the main purpose of the weak fairness is to compensate for the interleaving representation of concurrency, and ensure that every *process* will eventually progress (provided it is enabled, of course). Therefore, we wanted our fairness requirements to ensure this progress, but nothing more. In particular, we did not want the requirements to automatically imply *fairness in selection*, meaning that if a non-deterministic choice in one of the processes is repeated infinitely many times, then the process must choose fairly between the transitions involved in the choice. This is not because we are against fairness in selection, but rather that we prefer to separate the two issues, and have separate requirements that will represent process progress and nothing more, and another set that will represent fairness in selection. This leaves he designer of the programming language (in particular its semantics) the option to adopt one type of fairness without the other, or to adopt both.

To illustrate why the simpler $(\{\tau\}, \{\tau\})$ implies fairness in selection, while our requirement $(\{\tau\}, Comp(\tau))$ does not, consider the following simple program.

$$\{\ y = 1\ \}$$




$-\ P_1\ -$ $\qquad\qquad$ $-\ P_2\ -$

This program consists of two processes, $P_1$ and $P_2$. Process $P_2$ continuously waits for the variable $y$, which initially is positive, to become 0. The process $P_1$ has two transitions $\tau_1$ and $\tau_2$ in tight loops around the location $l_0$. The transition $\tau_1$ sets the value of $y$ to 0. The transition $\tau_2$ doubles the value of $y$. Note that once $y$ becomes 0, the transition $\tau_2$ preserves this value, so eventually $P_2$ will sense that $y = 0$ and proceed to location $m_1$.

The question is: is it guaranteed that in all computations of this program, eventually $\tau_1$ will be taken, causing $P_2$ to eventually reach $m_1$? In other words, is the following sequence of states (each state presented by listing the current values of $\pi_1, \pi_2, y$, respectively):

$$\sigma : \langle l_0, m_0, 1\rangle, \langle l_0, m_0, 2\rangle, \langle l_0, m_0, 4\rangle, ...,$$

obtained by continuously taking only the transition $\tau_2$, acceptable as a computation of the above program?

The answer depends on the fairness requirements we adopt, in particular the ones concerned with $\tau_1$. The weak fairness requirement associated with $\tau_1$, as defined in our semantics for the shared variable model, is given by $(\{\tau_1\}, \{\tau_1, \tau_2\})$. Under this requirement, the sequence above is weakly fair with respect to $\tau_1$, since even though it is continuously enabled and never taken, one of its competitors, $\tau_2$, is taken infinitely many times. Obviously this sequence is also weakly fair with respect to $\tau_2$, which is taken infinitely many times, and also to $\tau_3$, which is never enabled. Thus, according to our official semantics this sequence is admissible as a computation.

If, on the other hand, we take instead the naive requirement $(\{\tau_1\}, \{\tau_1\})$, then the sequence above is not weakly fair with respect to this requirement. This means that the naive requirement forces the process $P_1$ to be fair in its selection, and eventually take $\tau_1$.

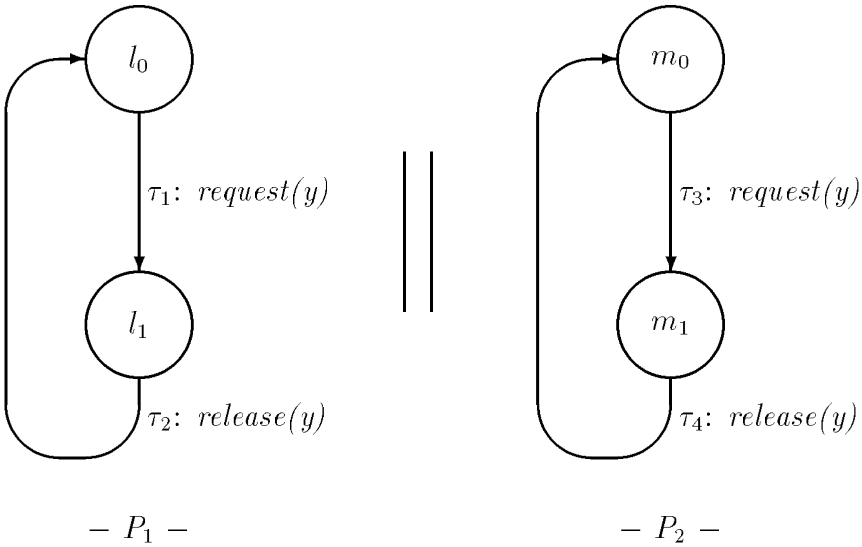
## 2.3 Semaphores

For additional synchronization and coordination between processes we introduce two special instructions, *request*$(y)$ and *release*$(y)$, called *semaphore* instructions. Operationally, they perform the same transition (and therefore have the same transition assertion $\rho_\tau$) as the guarded commands:

$$\begin{aligned} request(y) &: (y > 0) \rightarrow [y := y + 1] \\ release(y) &: y := y + 1 \end{aligned}$$

To illustrate their use, consider the following program:

$$\{\ y = 1\ \}$$




This program uses semaphores in order to coordinate mutual exclusion between the locations $l_1$ and $m_1$, i.e., to ensure that at no state in the computation can $P_1$ be at $l_1$, while at the same time, $P_2$ is at $m_1$. This is ensured by this program, because the first process to enter one of those *critical* locations sets $y$ to 0, which disbales the *request*$(y)$ of the other process, until the first process exits its critical location and resets $y$ to 1.

However, another property we would expect from a good implementation of a mutual exclusion program is *accessibility*. This property requires that any process that wishes to enter its critical location will eventually do so. In the simpler program above, the processes continuously wish to enter their critical location (as represented by the fact that the non-critical location has only one exit), and therefore accessibility means that each process should visit its critical location infinitely many times. Unfortunately, this property is not guaranteed under our current fairness assumptions. Consider for example the sequence of states:

$$\sigma : \langle l_0, m_0, 1\rangle, \langle l_1, m_0, 0\rangle, \langle l_0, m_0, 1\rangle, \langle l_1, m_0, 0\rangle, \langle l_0, m_0, 1\rangle, ...,$$

obtained by taking only transitions of $P_1$. This sequence satisfies all our fairness requirements. It is certainly weakly fair to $\tau_1, \tau_2$, which are taken infinitely many times. It is also trivially fair to $\tau_4$, which is never enabled. What about $\tau_3$? While it is enabled infinitely many times, at all occurrences of the state $\langle l_0, m_0, 1\rangle$, it is also disabled infinitely many times, at all occurrences of the state $\langle l_1, m_0, 0\rangle$. Thus, $\tau_3$ is not continuously enabled at any point, and hence the sequence above is also fair to $\tau_3$. It follows that this sequence is a computation, and hence the property of accessibility does not hold for this program.

To remedy this situation we extend our fairness requirements as follows:

- For each transition $\tau$ associated with a *request* instruction, we add the strong fairness requirement $(\{\tau\}, Comp(\tau))$ to the strong fairness family $\mathcal{F}$.

Note that there is no need to make a similar requirement for the *release* instruction, since when the control is in the location which is the source for this transition, the transition is always enabled. The requirement of strong fairness imposed above makes the sequence $\sigma$, considered previously, inadmissible as a computation, since it is not strongly fair with respect to the transition $\tau_3$, which is enabled infinitely many times but never taken. It is not difficult to see that, with the strong fairness requirements imposed for both $\tau_1$ and $\tau_3$, every computation must visit both $l_1$ and $m_1$ infinitely many times.

## 2.4 Concrete Model II : Synchronous Message Passing

As our next concrete model, we take a programming language in which communication between processes is performed by synchronous (hand-shaking) message passing, similar to CSP or CCS. For describing programs we may still use the diagram language, but enrich its repertoire of instructions that may label edges in the diagrams, by two new instructions:

- **Send**

  $$c \rightarrow [\alpha \Leftarrow e]$$

  This instruction sends on channel $\alpha$ the current value of the expression $e$. The enableness of this instruction is, in general, conditional on a boolean expression $c$

- **Receive**

  $$c \rightarrow [\alpha \Rightarrow y]$$

  This instruction reads from channel $\alpha$ a value into the variable $y$. Its execution is conditional on the boolean condition $c$.

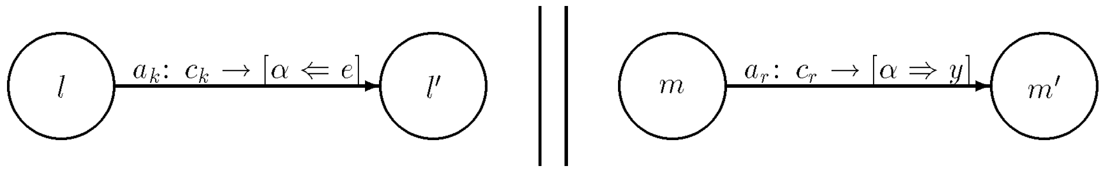
According to the synchronous communication protocol, a *receive* instruction in one process can be executed only jointly with a *send* instruction of a parallel process, which addresses the same channel. The joint execution causes both processes to advance across their respective edges, while the value of $e$ evaluated in the sending process is assigned to the variable $y$ in the receiving process. Consequently, we define the set of transitions $\mathcal{T}$ for a synchronously communicating program as follows:

- **Assignment**

  With each edge labeled by an assignment instruction, we associate a transition whose transition assertion is defined identically to the shared variable case.

- **Communication**

  Consider a *pair* of edges $a_k, a_r$ belonging to two parallel processes, $P_i, P_j$, respectively, which are labeled by a send and receive instructions, addressing the same channel, i.e.,




$-\ P_i\ -$ $\qquad\qquad$ $-\ P_j\ -$

We refer to such a pair as a *matching* pair of communication instructions. For each such pair we define a transition $\tau_{kr}$, whose transition relation is given by:

$$\rho_{\tau_{kr}} : (\pi_i = l) \wedge (\pi_j = m) \wedge c_k \wedge c_r \wedge (\pi_i' = l') \wedge (\pi_j' = m') \wedge (y' = e) \wedge \bigwedge_{u \in V - \{\pi_i, \pi_j, y\}} (u' = u)$$

Joint transitions, such as the ones considered above, have more than one source. For example, the transition $\tau_{kr}$ has as sources the locations $l$ in $P_i$, and $m$ in $P_j$. Consequently it has several *competition sets*, one associated with each source, or each process to which the transition belongs. As before we denote by $Comp(\tau)$ the *family* of competitions sets of the transition $\tau$. For the case that the transition $\tau$ is an assignment, the family $Comp(\tau)$ contains only a single competition set.

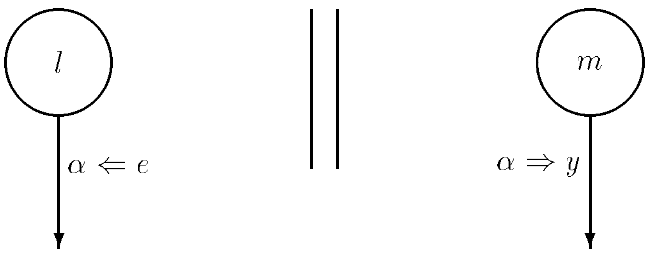
We formulate now the fairness requirements associated with synchronously communicating programs.

- Weak Fairness. For each transition $\tau$ and each competition set $\mathcal{C} \in Comp(\tau)$, we include in $\mathcal{W}$ the requirement $(\{\tau\}, \mathcal{C})$.

- Strong Fairness. For each *communication* transition $\tau$ and each competition set $\mathcal{C} \in Comp(\tau)$, we include in $\mathcal{F}$ the requirement $(\{\tau\}, \mathcal{C})$.

Thus, we associate strong fairness requirements only with the communication transitions, interpreting them as coordination transitions similar to semaphores. Let us examine the implications of these definitions.

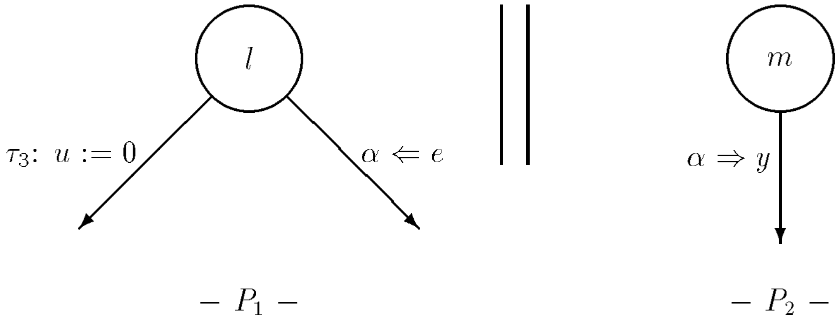
Consider first the following situation:




$-\ P_1\ -$ $\qquad\qquad$ $-\ P_2\ -$

In this case we have two processes with two matching communication instructions. However, none have competitors (we assume that the only edges departing from $l$ and $m$, are those depicted in the diagram). Consequently, the competition family of $\tau_{12}$ consists of a single set $\{\tau_{12}\}$. It follows that if a computation ever arrives to a state in which control is both at $l$ and $m$, then weak fairness alone is sufficient to guarantee that $\tau_{12}$ will eventually be taken. This is because, otherwise, it will be continuously enabled and never taken.

Next consider the following situation.



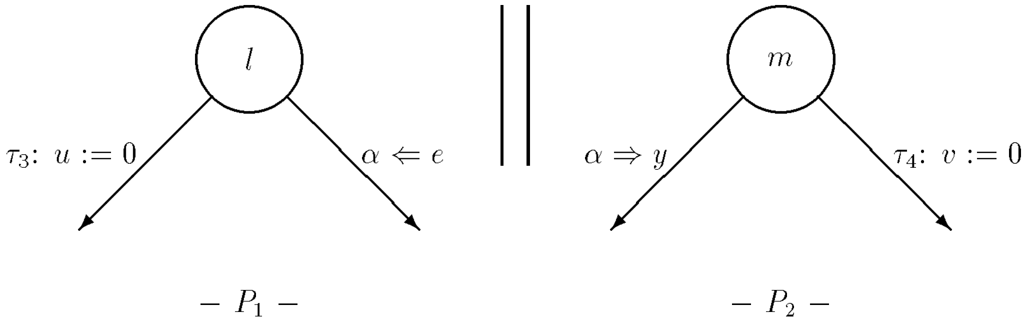

Here there are two alternatives departing from $l$, the communication and also another transition, which being a local assignment, is always enabled. Consider the situation in which $P_2$ arrives at $m$ and waits there for a communication, while $P_1$ visits $l$ infinitely often, always exiting via $\tau_3$. Is such a behavior allowed by our fairness requirements?

Let us examine the competition family of $\tau_{12}$. It is given by

$$Comp(\tau_{12}) = \{\{\tau_{12}, \tau_3\}, \{\tau_{12}\}\}$$

As we observe, it consists of two sets: $\{\tau_{12}, \tau_3\}$ – the competition set in $P_1$, and $\{\tau_{12}\}$ – the competition set in $P_2$. The behavior described above is strongly fair with respect to the requirement $(\{\tau_{12}\}, \{\tau_{12}, \tau_3\}$, since the transition $\tau_3$ is taken infinitely many times. However, this behavior is not strongly fair with respect to the second fairness requirement associated with $\tau_{12}$, namely $(\{\tau_{12}\}, \{\tau_{12}\}\}$. This is because $\tau_{12}$ is not taken even once. Thus, our fairness requirements guarantee that, in a situation such as above, where $P_2$ steadily waits at $m$, while $P_1$ visits $l$ infinitely many times, the communication will eventually take place.

Next consider a third situation.




Here, both processes have local alternatives to the joint transition. Assume a behavior containing infinitely many states in which both $P_1$ is at $l$ while $P_2$ is at $m$, but yet not taking the joint transition even once. This of course means that both processes consistently choose the local alternative. Does such a behavior satisfy our fairness requirements? Yes, because the two competition sets are $\{\tau_3, \tau_{12}\}$ and $\{\tau_{12}, \tau_4\}$, respectively, and each of them contains a transition that is taken infinitely many times.

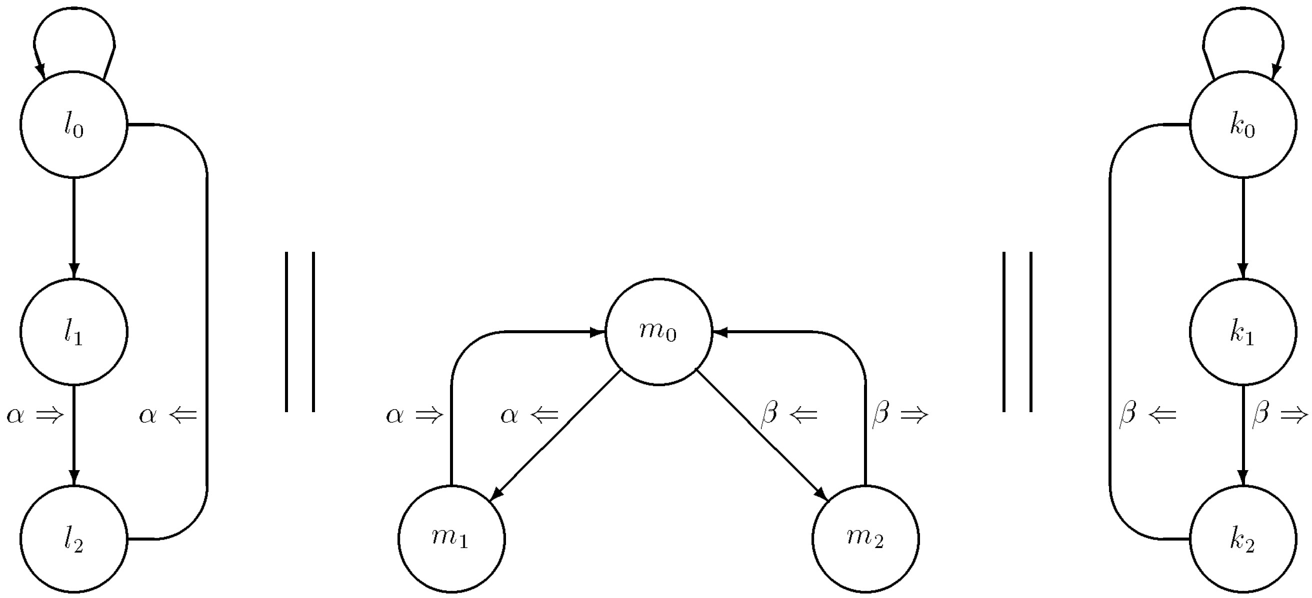
One may wonder whether the distinction we make between the two last cases is not somewhat arbitrary. We claim it is not. There is a certain degree of robustness we would like our definitions and concepts to have. For example, they should be robust against interchanging the order of two adjacent transitions which are independent of one another, i.e., belong to different processes and do not access the same variable or channel. Any behavior of the last example in which the processes jointly visited the locations $l, m$ infinitely many times, can be modified by such interchanges into an equivalent behavior in which $P_1$ is never at $l$ at the same time that $P_2$ is at $m$. This is because the entry transitions to this location in one of the processes must be independent of the exit local exit transition in the other. Consequently, we can make sure that one of the processes leaves the critical position before the other enters his. Clearly in such a modified behavior the question of the joint transition never arise, because it is never enabled. We claim that this should be the case with the original behavior. In such a situation we consider it quite coincidental that they visited their respective critical locations (i.e., $l$ and $m$) together.

The situation is quite different for the example where one of the processes is frozen in a waiting position, while the other visits its critical location infinitely often. Any interchanged behavior would retain the same characterization. We can make the entries and exits from the locations of the active process come a little earlier or later, but we cannot alter the fact that the other process is steadly waiting.

We can summarize this discussion by saying that the form of strong fairness we have adopted for synchronously communicating processes has the following characterization:

> If a communication transition between two processes is enabled infinitely often, then it is guaranteed to be taken only if one of the processes waits steadily, while the other may wander in and out of the communication location.

We conclude this discussion by showing how this approach can be used to implement mutual exclusion between two processes, using synchronous communication.




– $P_1$ –   – *Arbiter* –   – $P_2$ –

This program uses an arbiter process to mediate between the rivals $P_1$ and $P_2$. The arbiter synchronizes with $P_1$ by communicating along channel $\alpha$, and with $P_2$ along channel $\beta$. The reason that process $P_1$, for example, if it waits at $l_1$ long enough is guaranteed to eventually perform its communication is that $P_1$ in this case is the steady waiter, while the arbiter may wander in and out of $m_0$ several times. If $P_1$ had an internal transition allowing it to change its mind and go from $l_1$ back to $l_0$, then this guarantee would not have held any longer.

# 3 Temporal Logic

We assume an underlying assertional language, which contains at least the predicate calculus, and interpreted symbols for expressing the standard operations and relations over the integers. We refer to a formula in the assertion 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 $\quad \ominus$ – Previous
$\mathcal{U}$ – Until $\quad \mathcal{S}$ – Since

A *model* for a temporal formula $p$ is a finite or infinite sequence of states

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

where each state $s_j$ provides an interpretation for the variables mentioned in $p$. If $\sigma$ is finite, i.e., $\sigma : s_0, s_1, ..., s_k$, we define the *length* of $\sigma$ to be the number of states in $\sigma$, i.e., $|\sigma| = k + 1$. If $\sigma$ is infinite, we write $|\sigma| = \omega$.

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

- For a state formula $p$
$$(\sigma, j) \models p \iff s_j \models p.$$

For the case that $p$ is a state formula it can be evaluated locally, using the interpretation given by $s_j$ to the free variables appearing in $p$.

- $(\sigma, j) \models \neg p \quad \iff \quad (\sigma, j) \not\models p$
- $(\sigma, j) \models p \vee q \quad \iff \quad (\sigma, j) \models p$ or $(\sigma, j) \models q$
- $(\sigma, j) \models \bigcirc p \quad \iff \quad j + 1 < |\sigma|$ and $(\sigma, j + 1) \models p$
- $(\sigma, j) \models p \mathcal{U} q \quad \iff \quad$ for some $k, j \leq k < |\sigma|, (\sigma, k) \models q$, and for every $i, j \leq i < k, (\sigma, i) \models p$
- $(\sigma, j) \models \ominus p \quad \iff \quad j > 0$ and $(\sigma, j - 1) \models p$
- $(\sigma, j) \models p \mathcal{S} q \quad \iff \quad$ for some $k, 0 \leq k \leq j, (\sigma, k) \models q$, and for every $i, j \geq i > k, (\sigma, i) \models p$

Additional temporal operators can be defined as follows:

| | | | |
|---|---|---|---|
| $\widetilde{\bigcirc} p = \neg \bigcirc \neg p$ | – Weak Next | $\widetilde{\ominus} p = \neg \ominus \neg p$ | – Weak Previous |
| $\diamondsuit p = \mathbf{T} \mathcal{U} p$ | – Eventually | $\diamondminus p = \mathbf{T} \mathcal{S} p$ | – Sometimes in the past |
| $\square p = \neg \diamondsuit \neg p$ | – Henceforth | $\boxminus p = \neg \diamondminus \neg p$ | – Always in the past |
| $p \mathcal{W} q = \square p \vee p \mathcal{U} q$ | – Unless, Weak Until | $p \mathcal{B} q = \boxminus p \vee p \mathcal{S} q$ | –Weak Since |

Two additional useful derived operators are the *entailment* operator, defined by:

$$p \Rightarrow q \iff \square(p \rightarrow q),$$

and the *strong equivalence* operator, defined by

$$p \Leftrightarrow q \iff \square(p \equiv q).$$

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, to which we also refer as an *assertion*, is both a past and a future 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 $p \approx q$, if the formula $p \equiv q$ is valid, i.e., $\sigma \models p$ iff $\sigma \models q$.

The notion of validity defined above is the notion of *temporal validity*, i.e., a temporal formula holding at position 0 of all possible sequences of states. A simpler notion we need now and then is that of *assertional validity*, which is that of an assertion $p$ holding over every possible state. When we have to discuss the two together, we denote the temporal validity of a formula $p$ by $\models_T p$, and the assertional validity of an assertion $q$ by $\models_A q$. Obviously, an assertion $q$ is assertionally valid iff it is temporally valid, because any state can be used as the first state in a sequence. That is:

$$\models_A q \iff \models_T q.$$

For the use of temporal formulae for specifying the properties of programs, we need to restrict the two notions of validity to a smaller sets of models, i.e., the models corresponding to computations of a given program. First we will give a general definition for an arbitrary set of models. Let $\mathcal{C}$ be a set of models, i.e., a set of finite or infinite sequences of states. We define a temporal formula to be (temporally) $\mathcal{C}$-valid, denoted by $\mathcal{C} \models_T p$ (or simply $\mathcal{C} \models p$, when there is no danger of confusion), if $p$ holds at the first position of every sequence of $\mathcal{C}$. Similarly, we define an assertion $q$ to be (assertionally) $\mathcal{C}$-valid, denoted by $\mathcal{C} \models_A q$, if $q$ holds at all the states appearing at any position in the sequences of $\mathcal{C}$. Note that under a restriction to some subset $\mathcal{C}$, it is no longer the case that an assertion $q$ is assertionally valid iff it is temporally valid, i.e.,

$$\neg(\mathcal{C} \models_A q \iff \mathcal{C} \models_T q)$$

This is because, in a restricted set of models, the states appearing at the first position in sequences may be a strict subset of the states appearing anywhere in the sequences of $\mathcal{C}$.

Given a program $P$, we can restrict our attention to the set of models which correspond to computations of $P$, i.e., $\mathcal{C}(P)$. This leads to the corresponding notions of validity over $P$. Thus, for a temporal formula $p$, we say that $p$ is (temporally) $P$-valid, denoted by $P \models_T p$, if $p$ holds at the first position of all the computations of $P$. For an assertion $q$, we say that $q$ is (assertionally) $P$-valid, denoted by $P \models_A q$, if $q$ holds over all *accessible* states of $p$, i.e., states appearing at some position in a computation of $P$. Similarly, we obtain the notions of $P$-satisfiability and $P$-equivalence.

# 4 The Classification of Properties

Having defined the notions of programs and their computations, which are sequences of states, and how temporal formulae are interpreted over sequences, it is obvious how a temporal formula may serve as a specification for a program. A program $P$ is said to satisfy (or implement) the specification $\varphi$, if $\varphi$ is $P$-valid. In general, a specification is structured as a conjunction of *requirements*, each of them describing some property that the program is supposed to have. One of the main advantages of this conjunctive structure of the specification is its *incrementality*. This means that if, at a later stage we wish to add, delete, or modify, one of the requirements, this can be done by dealing with this requirement alone, without directly interfering with any of the others. Similarly, in order to verify that a proposed implementation satisfies the specification, it is sufficient to verify each of the properties separately. In that, it is superior to some of the alternative specification methods, such as those based on process algebra formalisms, and provide an abstract modeling of the specified system. There the model integrates all the requirements together, and any modification of one of them, may require a major overhaul of the complete model.

However, there is another aspect, in which the abstract modeling approach is superior to the logic based approach, and this is the question of completeness of the specification. In the conjunctive form of the specification, there is always the question whether we have included all the properties that are relevant to the program we wish to specify. As we indicated in the introduction, one of the ways to try to ensure completeness of the specification is to classify the properties into several classes of frequently occurring properties. Then, when we contemplate a specification, we can at least compare it against a check list of types of properties, and for each class ask whether we have included a property of this type in the specification, or perhaps recognize that this class is irrelevant to the considered program.

This is the reason why there is a great interest in structuring the universe of expressible properties into several classes, each of which typically represents a certain type of phenomena, and is associated with a a unique style of verification needed for its establishment.

The first such classification of properties into the classes of *safety* and *liveness* properties has been suggested by Lamport in [Lam77]. The definition of the two classes has been given informally, by the descriptions:

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

Examples of safety properties are those which are expressible by an invariance formula of the form $\Box\, p$, and state that some state property $p$ continuously holds on all states in the computation of the program. For example, $p$ may state that mutual exclusion is maintained in the current state, i.e., that if one of the processes is critical then the other is not. Then, $\Box\, p$ claims that mutual exclusion always holds.

Examples of liveness properties are those which are expressible by an eventuality formula of the form $\Diamond\, q$, and state that some state property will eventually hold on each computation of the program. For example, if $q$ designates termination, then $\Diamond\, q$ claims that the program always terminates.

As has been established over the years, this partition also distinguishes between two distinct proof approaches which are needed to establish the validity of properties in the two classes. To prove safety properties one uses essentially *computational induction* (using the terminology of [Man74]), based on an invariance principle. According to this principle, one shows that $p$ holds in the initial state, and then, that $p$ is preserved by any transition of the program. Consequently, based on an induction argument on the position in the computation, it follows that $p$ holds at all positions. 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., that $p$ holds initially, and that it is preserved over any transition. Thus the induction is *implicit*.

Liveness properties, on the other hand, are proven using *structural induction* (using the terminology of [Man74]), i.e., explicit induction on some element of the state. This induction is often represented as an application of a well-founded argument. The reason we need explicit induction for liveness properties, such as $p \rightarrow \Diamond\, q$, is that when the program starts at a $p$-state, there are in general, several important intermediate stages that the computation has to evolve through on the way from $p$ to $q$. The well-founded argument is used to measure how close we are to the goal $q$. 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 [Lam83b], and a semantic formalization 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 formulae $\mathbf{T}, \mathbf{F}$).
- Every property can be represented as the intersection of a safety property and a liveness property.
- The classes contain the obvious properties that are informally associated with them, i.e., $\Box\, p$ in the safety class, and $\Diamond\, q, p \rightarrow \Diamond\, q, \Box \Diamond\, q$, etc. in the liveness class, for state formulae $p, q$.
- 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 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, as the set of formulae constructed from state formulae to which we apply the positive boolean operations $(\wedge, \vee)$, and the temporal operator $\mathcal{W}$ (*Unless*). Note that $\Box$ is a special case of $\mathcal{W}$, since $\Box\, p = p \mathcal{W} \mathbf{F}$. He also gave some characterizations of some subclasses of the liveness class, which however do not cover the full class. Another problem with these characterizations, which are all done within the future fragment, is that they do not lead directly to a useful proof principle.

An intersting development occurred when, incorporating the past fragment into the logic, it was realized ([LPZ85]) that the full class of safety properties expressible in temporal logic, can be expressed by a formula of the form $\Box\, p$ for some *past* formula $p$. This seems a most natural generalization of the classical example of safety formulae which have the form $\Box\, p$ for a *state* formula $p$. Based on this observation, we have developed a small hierarchy of classes of properties, of which safety is the lowest class. Unfortunately, while agreeing with the safety-liveness classification on the safety class, we do not have any class in the hierarchy which corresponds to the liveness class, which makes our classification somewhat orthogonal to the safety-liveness partition.

In the current section we present our classification of the temporal properties, to which we refer as the Borel hierarchy, since it coincides with the two lower levels of the Borel topological hierarchy. We examine this classification from three points of view. The first is semantic or topological, in which we characterize the classes as sets of sequences. Next, we consider properties which are expressible in temporal logic, and for those give for each class a syntactic characterization of the formulae which express properties falling in this class. Finally we consider the expression of properties by predicate automata, and give syntactic characterization of the automata describing properties falling in each class. We show that these three view coincide, i.e., any property expressible by temporal logic, which is semantically classified as belonging to class $\kappa$, is expressible by a $\kappa$-type formula. A similar result holds for properties expressible by automata. In the last section dealing with the program part of the proof system, we will indicate what are the proof principles associated with each of the classes.

## 4.1 Semantic View

We introduce the following notation. Let $\Sigma$ denote the set of states. We denote by $\Sigma^+$ the set of all finite but non empty sequences of states, and by $\Sigma^\omega$ the set of all infinite state sequences. Let $\Sigma^\infty = \Sigma^+ \cup \Sigma^\omega$ denote the set of both finite and infinite sequences. For a finite sequence $\sigma \in \Sigma^+$ and a sequence $\sigma' \in \Sigma^\infty$, we denote by $\sigma \prec \sigma'$ the fact that $\sigma$ is a finite prefix of $\sigma'$ but different from $\sigma'$ (a proper finite prefix). We denote by $\sigma \preceq \sigma'$ the more general relation $(\sigma \prec \sigma') \vee (\sigma = \sigma' \in \Sigma^+)$. We denote by $\sigma \cdot \sigma'$ the sequence, obtained by concatenating $\sigma'$ to the end of $\sigma$. Note that the concatenation is defined only if $\sigma$ is finite.

We (semantically) define a property to be a set $\Pi \subseteq \Sigma^\infty$. Intuitively, since we have no language to express the requirements the property should fulfill, the only way we can describe a property is by the set of all sequences that have this property. A property is called *finitary* if it contains only finite sequences (may still be an infinite set of such sequences), and it is called an *infinitary* property if it only contains infinite sequences. For a property $\Pi \subseteq \Sigma^\infty$, we denote by $Pref(\Pi)$ the set of all *finite* prefixes of $\Pi$:

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

We denote by $\sigma[0..k]$ the finite prefix $s_0, s_1, ..., s_k$ of the sequence $\sigma : s_0, s_1, ..., s_k, s_{k+1}, ...$.

The intuition behind the classification is based on a loose generalization of the basic informal characterization given by Lamport for the two classes. We distinguish four different modalities that can describe the frequency of occurrences of "good things" in a sequence. Informally:

- A *safety* property states that a good thing happens at *all* states of the sequence.
- A *termination* property states that a good thing happens at *some* states of the sequence.
- A *recurrence* property states that a good thing happens at *infinitely many* states of the sequence.
- A *persistence* property states that a good thing happens at *all but finitely many* states of the sequence.

It can be claimed that the above definition makes sense only if we can check at any state of the sequence, whether the good thing holds there or not. Consequently, we identify the good thing as a general property of finite prefixes of a sequence, i.e., as a finitary property. Thus, we can view the classification as based on the different ways we can derive an infinitary property from a finitary one.

To formalize these notions, we introduce four operators $A, E, R$, and $P$, which map finitary properties on properties contained in $\Sigma^\infty$. They are defined as follows:

- $\sigma \in A(\Pi) \iff \forall \sigma'(\sigma' \preceq \sigma) : \sigma' \in \Pi.$

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

- $\sigma \in E(\Pi) \iff \exists \sigma'(\sigma' \preceq \sigma) : \sigma' \in \Pi$

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

- $\sigma \in R(\Pi) \iff \forall \sigma'(\sigma' \preceq \sigma) : \exists \sigma''(\sigma' \preceq \sigma'' \preceq \sigma) : \sigma'' \in \Pi.$

  Obviously, $\sigma \in R(\Pi)$ iff

  either $\sigma$ is finite and belongs to $\Pi$,
  or $\sigma$ is infinite and *infinitely many* finite prefixes of $\sigma$ are in $\Pi$.

- $\sigma \in P(\Pi) \iff \exists \sigma'(\sigma' \preceq \sigma) : \forall \sigma''(\sigma' \preceq \sigma'' \preceq \sigma) : \sigma'' \in \Pi$.

  Obviously, $\sigma \in P(\Pi)$ iff

  either $\sigma$ is finite and belongs to $\Pi$,
  or $\sigma$ is infinite and *all but finitely many* finite prefixes of $\sigma$ are in $\Pi$.

For $X$, one of the four operators $A, E, R, P$, we define the finitary and infinitary versions of $X$ by:

$$X_f(\Pi) = X(\Pi) \cap \Sigma^+ \quad \text{and} \quad X_\omega(\Pi) = X(\Pi) \cap \Sigma^\omega$$

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

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

We refer to these four classes of properties, denoted by $\mathcal{S}, \mathcal{T}, \mathcal{R}, \mathcal{P}$, respectively, as the *basic* classes. In addition we have four more classes that can be constructed by boolean combinations of the basic classes.

We define a property $\Pi$ to be an *intermittence* property, belonging to the class $\mathcal{I}$,if

$$\Pi = A_\omega(\Pi') \cup E_\omega(\Pi''),$$

for some finitary $\Pi'$ and $\Pi''$.

We define a property $\Pi$ to be a *multiple intermittence* property, belonging to the class $\mathcal{MI}$, if it can be obtained by a finite boolean combination, i.e., union, intersection, and complementation (relative to $\Sigma^\omega$) of safety and termination properties.

We define a property $\Pi$ to be a *progress* property, belonging to the class $\mathcal{G}$, if

$$\Pi = R_\omega(\Pi') \cup P_\omega(\Pi''),$$

for some finitary $\Pi'$ and $\Pi''$.

We define a property $\Pi$ to be a *multiple progress* property, belonging to the class $\mathcal{MG}$, if it can be obtained by a finite boolean combination of recurrence and persistence properties.

The motivation for these names will be given in the next subsection.

We observe the following facts about the defined classes.

**Fact 4.1 Duality:** *The classes $\mathcal{S}$ and $\mathcal{T}$ are dual under complementation, i.e., $\Pi \subseteq \Sigma^\omega$ is a* **safety** *property iff $\Sigma^\omega - \Pi$ is a* **termination** *property. Similarly, the classes $\mathcal{R}$ and $\mathcal{P}$ are dual. The classes $\mathcal{MI}$ and $\mathcal{MG}$ are closed under complementation.*

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

$$\begin{aligned} \Sigma^\omega - A_\omega(\Pi) &= E_\omega(\Sigma^+ - \Pi), \quad \text{and} \\ \Sigma^\omega - E_\omega(\Pi) &= A_\omega(\Sigma^+ - \Pi) \end{aligned}$$

This is because

$$
\begin{array}{rcl}
\sigma \in (\Sigma^\omega - A_\omega(\Pi)) & \leftrightarrow & \sigma \notin A_\omega(\Pi) \\
& \leftrightarrow & \neg(\forall \sigma'(\sigma' \preceq \sigma) : \sigma' \in \Pi) \\
& \leftrightarrow & \exists \sigma'(\sigma' \preceq \sigma) : \sigma' \notin \Pi \\
& \leftrightarrow & \exists \sigma'(\sigma' \preceq \sigma) : \sigma' \in (\Sigma^+ - \Pi). \\
& \leftrightarrow & \sigma \in E_\omega(\Sigma^+ - \Pi)
\end{array}
$$

Similarly:

$$
\begin{array}{rcl}
\Sigma^\omega - R_\omega(\Pi) & = & P_\omega(\Sigma^+ - \Pi), \quad \text{and} \\
\Sigma^\omega - P_\omega(\Pi) & = & R_\omega(\Sigma^+ - \Pi)
\end{array}
$$

The class $\mathcal{MI}$, being defined as a boolean combination of safety and termination properties, is certainly closed under an additional boolean operation such as complementation. The same holds for the class $\mathcal{MG}$.

**Fact 4.2 Closure:** *The classes $\mathcal{S}, \mathcal{T}, \mathcal{MI}, \mathcal{R}, \mathcal{P}, \mathcal{MG}$ are closed under union and intersection. The classes $\mathcal{I}, \mathcal{G}$ are closed under union.*

To show these closure properties, we need some operations on *finitary* properties. We use freely the boolean operations of union and intersection, and complementation with respect to $\Sigma^+$. we also use concatenation of properties:

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

the special property $\Sigma$ is the set of all singleton sequences, i.e., sequences of length 1.
Note that using concatenation between properties, we can express the operator $E(\Pi)$ and its restricted versions, for a finitary $\Pi$, as:

$$E(\Pi) = \Pi \cdot \Sigma^\infty, \quad E_f(\Pi) = \Pi \cdot \Sigma^+, \quad E_\omega(\Pi) = \Pi \cdot \Sigma^\omega$$

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

$$(\Pi_1)\mathbf{S}(\Pi_2) \; = \{\sigma \in \Sigma^+ \mid \exists \sigma'(\sigma' \preceq \sigma) : [(\sigma' \in \Pi_2) \; \wedge \; (\forall \sigma''(\sigma' \prec \sigma'' \preceq \sigma) : \sigma'' \in \Pi_1)]\}.$$

According to this definition $\sigma \in (\Pi_1)\mathbf{S}(\Pi_2)$ iff $\sigma$ has a finite prefix $\sigma'$ in $\Pi_2$ and all other prefixes of $\sigma$, longer than $\sigma'$, are in $\Pi_1$.

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

- For *safety* they are justified by:

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

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

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

$\sigma'' \in A_f(\Pi_1) \cup A_f(\Pi_2)$. Hence, any finite prefix of $\sigma$ is in $A_f(\Pi_1) \cup A_f(\Pi_2)$. We conclude that $\sigma \in A_\omega(A_f(\Pi_1) \cup A_f(\Pi_2))$, and therefore:

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

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

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

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

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

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

- For *recurrence*, we claim:

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

Obviosuly, $\sigma$ contains either infinitely many $\Pi_1$-prefixes or infinitely many $\Pi_2$-prefixes iff it contains infinitely many $(\Pi_1 \cup \Pi_2)$-prefixes.

Closure under intersection is given by the equality:

$$R_\omega(\Pi_1) \cap R_\omega(\Pi_2) \;=\; R_\omega(\Pi_1 \cap [(\overline{\Pi_1})\mathbf{S}(\Pi_2)] \cdot \Sigma),$$

where we denote, for a finitary property $\Pi$, $\overline{\Pi} = \Sigma^+ - \Pi$.

We observe that $\sigma \in (\Pi_1 \cap [(\overline{\Pi_1})\mathbf{S}(\Pi_2)] \cdot \Sigma)$ iff $\sigma \in \Pi_1$, $\sigma$ has a prefix $\sigma' \prec \sigma$ such that $\sigma' \in \Pi_2$, and all other prefixes, longer than $\sigma'$ and shorter than $\sigma$, are not in $\Pi_1$. This characterizes a finite sequence $\sigma$ in $\Pi_1$, such that its longest proper prefix which belongs to $\Pi_1 \cup \Pi_2$ belongs in fact to $\Pi_2$.

Obviously, $\sigma$ has infinitely many $\Pi_1$-prefixes as well as infinitely many $\Pi_2$-prefixes iff it has infinitely many $\Pi_1$-prefixes whose longest proper $(\Pi_1 \cup \Pi_2)$-prefix is a $\Pi_2$-prefix.

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

$$\begin{aligned} P_\omega(\Pi_1) \cap P_\omega(\Pi_2) &= P_\omega(\Pi_1 \cap \Pi_2), \\ P_\omega(\Pi_1) \cup R_\omega(\Pi_2) &= P_\omega(\Pi_1 \cup \overline{[(\Pi_1)\mathbf{S}(\overline{\Pi_2})] \cdot \Sigma}). \end{aligned}$$

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

$$[A_\omega(\Pi_1) \cup E_\omega(\Pi_2)] \cup [A_\omega(\Pi_1') \cup E_\omega(\Pi_2')] \;=\; A_\omega(\Pi_1'') \cup E_\omega(\Pi_2''),$$

where
$A_\omega(\Pi_1'') = A_\omega(\Pi_1) \cup A_\omega(\Pi_1')$, guaranteed by the closure properties of $\mathcal{S}$, and
$E_\omega(\Pi_2'') = E_\omega(\Pi_2) \cup E_\omega(\Pi_2')$, guaranteed by the closure properties of $\mathcal{T}$

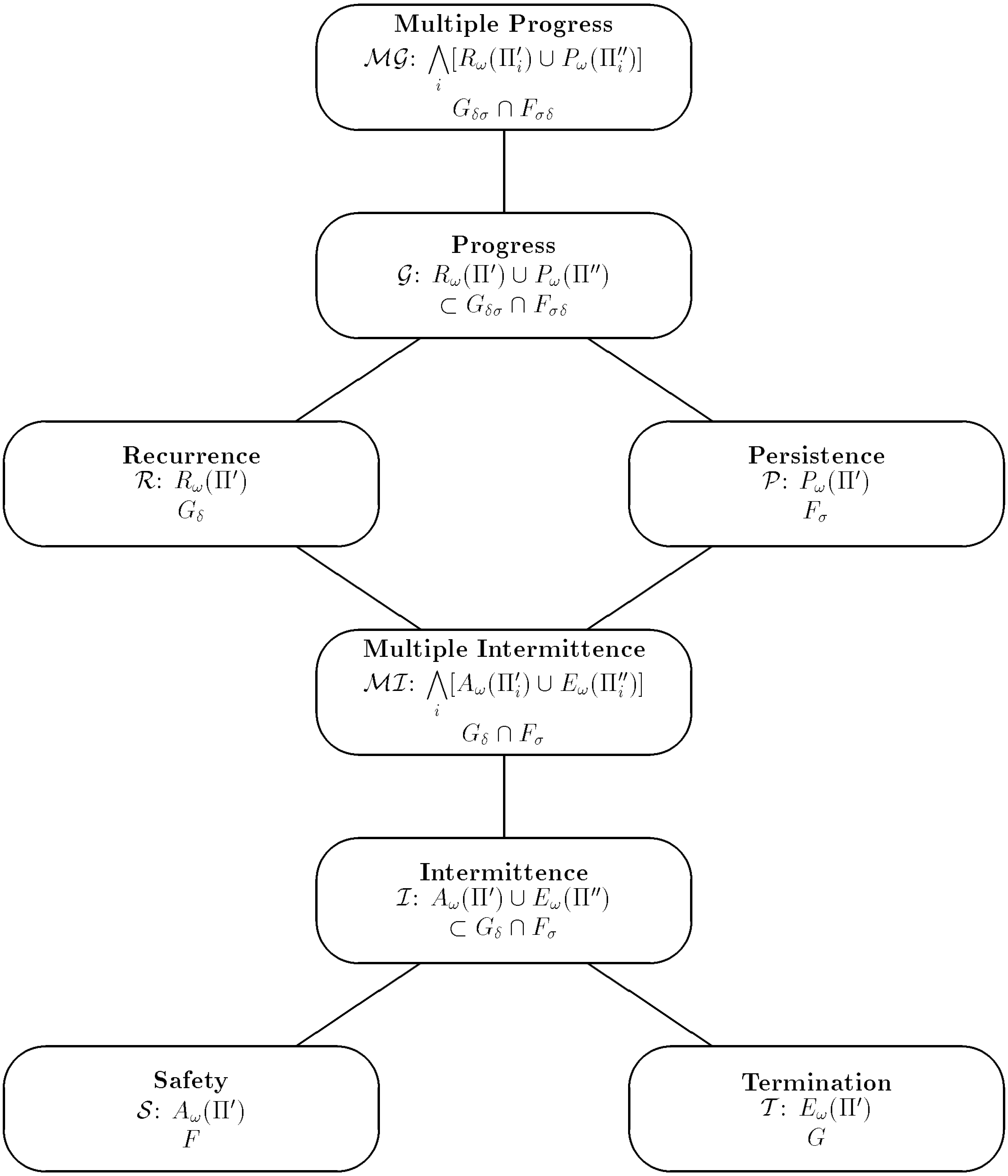



Figure 1: Inclusion Relations between the Classes

- The class of *progress* properties is handled similarly, using the closure under union of the classes $\mathcal{R}, \mathcal{P}$.
- The closure of $\mathcal{MI}, \mathcal{MG}$ is obvious.

**Fact 4.3 Inclusion:** *The classes of properties are related by the inclusion relations depicted in the diagram of Figure 1. The edges in the diagram represent strict inclusions.*

Note that the families of intermittence and progress properties are only subsets of the topological families $G_\delta \cap F_\sigma$ and $G_{\delta\sigma} \cap F_{\sigma\delta}$, respectively. The families of multiple intermittence and multiple progress, on the other hand, coincide precsicely with these topological families.
To show the inclusions $\mathcal{S} \cup \mathcal{T} \subseteq \mathcal{R}$ and $\mathcal{S} \cup \mathcal{T} \subseteq \mathcal{P}$, we observe that:

$$A_\omega(\Pi) = R_\omega(A_f(\Pi)) = P_\omega(A_f(\Pi)),$$

$$E_\omega(\Pi) = R_\omega(E_f(\Pi)) = P_\omega(E_f(\Pi)).$$

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

The other inclusions are equally easy to show. The strictness of the inclusions, not related to the classes $\mathcal{MI}$ and $\mathcal{MG}$ are also straightforward.

**Corollary 4.1 Normal Form:** *Any* multiple intermittence *property is expressible as the intersection of several* intermittence *properties. Similarly, any* multiple progress *property is expressible as the intersection of several* progress *properties.*

Thus, any *multiple intermittence* property $\Pi$ is expressible as:

$$\Pi = \bigcap_{i=1}^{k} (A_\omega(\Pi'_i) \cup E_\omega(\Pi''_i)),$$

for some finitary $\Pi'_i$ and $\Pi''_i, i = 1, ..., k$.

Similarly, any *multiple progress* property $\Pi$ is expressible as:

$$\Pi = \bigcap_{i=1}^{k} (R_\omega(\Pi'_i) \cup P_\omega(\Pi''_i)),$$

for some finitary $\Pi'_i$ and $\Pi''_i, i = 1, ..., k$.

Let $\Pi \subseteq \Sigma^\omega$ be a multiple intermittence property. By definition it can be expressed as a boolean combination of safety and termination properties. We perform the following transformations on the boolean expressions:

First we push all the complementations inside, using De Morgan laws. By the duality properties described in fact 4.1, all the complementations can be reduced to operations on finitary properties.

We are left with a positive boolean combination of safety and termination properties. We expand the resulting expression by distributivity to obtain a conjunctive normal form, i.e., an intersection of unions of safety and termination properties. By associativity of the union, we can group each union of safety properties and use the closure of the safety class under union to replace the union of safety properties by a single safety property. Similarly, we union together several termination properties into a single termination property. Thus we are left with an intersection of several clauses, each of which is a union of a single safety property and a single termination property. This leads directly to the normal form for the *multiple intermittence* class.

An identical argument leads to the normal form for the *multiple progress* properties.

We may in fact split the class of *multiple intermittence* properties into an infinite hierarchy, $\mathcal{I} = \mathcal{I}_1 \subset \mathcal{I}_2 \subset \ldots$, where $\mathcal{I}_k$ for $k \geq 1$ is defined as the class of all properties that can be expressed as the intersection of (at most) $k$ *intermittence* properties. It can be shown that this hierarchy is strict, i.e., for every $k$, there are properties which belong to $\mathcal{I}_{k+1}$ but not to $\mathcal{I}_k$. Obviously, $\mathcal{MI} = \bigcup_{k\geq 1} \mathcal{I}_k$, and the strictness of the hierarchy implies the strict inclusion of $\mathcal{I} = \mathcal{I}_1$ in $\mathcal{MI}$.

The hierarchy we study here is a *conjunctive* hierarchy, meaning that the outermost operator is an intersection or a conjunction. Previous discussions in the literature consider a *disjunctive* hierarchy of the *multiple progress* properties (at least those which are definable by an $\omega$-automaton). This hierarchy has been studied in [Lan69],[Wag79], and [Kam85], and is related to the Rabin's index of an $\omega$-automaton. The $k'$th layer in this hierarchy is given as:

$$\Pi = \bigcup_{i=1}^{k} (R_\omega(\Pi_i') \cap P_\omega(\Pi_i''))$$

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

## Comparison with the Safety–Liveness Partition

In this section we would like to compare our definition with the characterizations of safety and liveness as given in [AS85].

A safety property is characterized in [AS85] as a property $\Pi \subseteq \Sigma^\omega$ such that

$$\sigma \in \Pi \iff \forall \sigma'(\sigma' \preceq \sigma) : \exists \sigma''(\sigma'' \in \Sigma^\omega) : \sigma' \cdot \sigma'' \in \Pi \tag{1}$$

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

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

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

$$\sigma \in \Pi \iff \forall \sigma'(\sigma' \preceq \sigma) : \sigma' \in Pref(\Pi) \iff \sigma \in A_\omega(Pref(\Pi)).$$

This is the same as

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

It follows that any property satisfying (1) can be expressed as $A_\omega(\Pi')$, where $\Pi'$ happens to be $Pref(\Pi)$. It is not difficult to see that, if $\Pi = A_\omega(\Pi')$ for an arbitrary $\Pi'$, then in fact $\Pi$ can also be expressed as $A_\omega(Pref(\Pi))$.

Liveness is characterized in [AS85] as an infinitary property $\Pi \subseteq \Sigma^\omega$, such that

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

That is, every arbitrary finite sequence can be extended to an infinite sequence which belongs to $\Pi$.

As we already commented, the liveness class is orthogonal to our classification. For one, the liveness class is disjoint to the safety class. In our case we have an inclusive hierarchy, in which all the classes, excepting the termination class contain the safety class as a subclass. Because of the orthogonality of the two concepts, it is possible to inquire about properties which are $\kappa$-*live*, which means that they are both a liveness property and a property of class $\kappa \in \{\mathcal{T}, \mathcal{I}_k, \mathcal{R}, \mathcal{P}, \mathcal{G}_k\}$ in our hierarchy. In fact, for every property $\Pi$, of class $\kappa$, as defined above (i.e., in any class but the safety class), we can define the *liveness closure* of $\Pi$, denoted by $\mathcal{L}(\Pi)$, which is the smallest liveness property containing $\Pi$. In fact, the liveness closure of $\Pi$ is given by

$$\mathcal{L}(\Pi) = \Pi \cup E_\omega(\Sigma^+ - Pref(\Pi)).$$

and can be shown to be also of class $\kappa$. Clearly, $\sigma \in \mathcal{L}(\Pi)$ iff either $\sigma \in \Pi$ or $\sigma$ contains a prefix $\sigma' \prec \sigma$ that *cannot* be extended to a sequence in $\Pi$. Take any finite sequence $\sigma_0$. Either it can be extended to a sequence in $\Pi$, or it belongs to $\Sigma^+ - Pref(\Pi)$, in which case any infinite extension of it will belong to $E_\omega(\Sigma^+ - Pref(\Pi))$. This shows that $\mathcal{L}(\Pi)$ is a liveness property. Due to the closure properties of the classes $\mathcal{T}, \mathcal{I}, \mathcal{R}, \mathcal{P}, \mathcal{G}$, the property $\mathcal{L}(\Pi)$ belongs also to the class $\kappa$ to which $\Pi$ belongs.

One of the results of [AS85] is that each property is the intersection of a safety property and a liveness property. We can extend this result by claiming that every property of class $\kappa \in \{\mathcal{T}, \mathcal{I}_k, \mathcal{R}, \mathcal{P}, \mathcal{G}_k\}$ is the intersection of a liveness property of class $\kappa$ and a safety property. This is due to the equality

$$\Pi = \mathcal{L}(\Pi) \cap A_\omega(Pref(\Pi)).$$

## Topological Characterization

It is possible to locate the classes of properties in our hierarchy in the lower two (and a half) levels of the Borel topological hierarchy. The set $\Sigma^\omega$ can be made into a metric space by defining the distance between two sequences $\sigma$ and $\sigma' \in \Sigma^\omega$ to be

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

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

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

| | | | |
|---|---|---|---|
| $(\mathcal{S} = F)$ | $\Pi$ is a *safety* property | **iff** | it is a *closed* set |
| $(\mathcal{T} = G)$ | $\Pi$ is a *termination* property | **iff** | it is an *open* set |
| $(\mathcal{MI} = G_\delta \cap F_\sigma)$ | $\Pi$ is a *multiple intermittence* property | **iff** | it is a $G_\delta \cap F_\sigma$ set |
| $(\mathcal{R} = G_\delta)$ | $\Pi$ is a *recurrence* property | **iff** | it is a $G_\delta$ set |
| $(\mathcal{P} = F_\sigma)$ | $\Pi$ is a *persistence* property | **iff** | it is an $F_\sigma$ set |
| $(\mathcal{MG} = G_{\delta\sigma} \cap F_{\sigma\delta})$ | $\Pi$ is a *multiple progress* property | **iff** | it is a $G_{\delta\sigma} \cap F_{\sigma\delta}$ set |

In the above, we have denoted by $F$ the family of all closed sets, by $G$ the family of all open sets, by $G_\delta$ all sets obtainable as a countable intersection of open sets, and by $F_\sigma$ all sets obtainable by a countable union of closed sets.

For example, let us show that the class of termination properties coincides with the class of open sets.

- Let $\Pi = E_\omega(\Pi')$ be a termination set. Let $\sigma \in \Pi$ be any sequence in $\Pi$. By definition, $\sigma$ has a finite prefix $\sigma' \prec \sigma$ which belongs to $\Pi'$. Assume its length to be $k$. Then we claim that all infinite sequences whose distance from $\sigma$ does not exceed $2^{-(k+1)}$ are also in $\Pi$. Let $\sigma''$ be such an infinite sequence. Obviously, $\sigma$ and $\sigma''$ must agree on at least the first $k$ states. Consequently, $\sigma''$ also has $\sigma'$ as a finite prefix. Consequently, $\sigma'' \in \Pi$.

- Let $\Pi$ be an open set. Take any infinite sequence $\sigma \in \Pi$. Since $\Pi$ is open, there must be some distance $2^{-(k+1)}$ such that all infinite sequences closer to $\sigma$ than that distance are in $\Pi$. Let $\sigma'$ be the prefix of length $k$ of $\sigma$. We define $\Pi'$ to be the set of all such prefixes. It is not difficult to see that $E_\omega(\Pi') = \Pi$.

## 4.2 Expressiveness in Temporal Logic

Next, we restrict our attention to infinitary properties that can be expressed in temporal logic.

A temporal formula $p$ specifies a property $\Pi(p)$ given by:

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

Note that two formulae $p$ and $q$ are equivalent, denoted by $p \approx q$, iff $\Pi(p) = \Pi(q)$.

Since we are interested (for simplicity) in infinitary properties only, we define the *infinitary* property specified by the formula $p$ to be $\Pi_\omega(p) = \Pi(p) \cap \Sigma^\omega$. Alternately, we can define $\Pi_\omega(p) = \Pi(p \wedge \Box \bigcirc \mathbf{T})$, where $\Box \bigcirc \mathbf{T}$ specifies $\Sigma^\omega$. Corresponding to the restriction to infinite sequences, we define two formulae $p$ and $q$ to be $\omega$-equivalent, denoted by $p \approx_\omega q$, if $\Box \bigcirc \mathbf{T} \rightarrow (p \equiv q)$ is valid, i.e., $p$ is equivalent to $q$ on any infinite sequence. Obviously $p \approx_\omega q$ iff $\Pi_\omega(p) = \Pi_\omega(q)$.

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

- *Safety*

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

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

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

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

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

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

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

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

- *Termination*

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

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

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

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

Note that $p\mathcal{U}q$, for past formulae $p$ and $q$, expresses a termination property, since it is equivalent to:

$$p\mathcal{U}q \approx \Diamond(q \wedge \circleddash \boxminus p).$$

- *Intermittence*

A Formula of the form $(\Box p \vee \Diamond q)$ for some past formulae $p$ and $q$ is called an *intermittence* formula. An equivalent form for such formulae is give by the implication:

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

which states that if $p$ will eventually happen, then so will $q$. This type of statement is the basis for the intermittent assertions proof method ([MW78]).

Clearly, a multiple intermittence formula is a formula of the form:

$$\bigwedge_{i=1}^{k}(\Box p_i \vee \Diamond q_i),$$

which allows several intermittence conjuncts.

- *Recurrence*

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

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

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

$$(p \Rightarrow \Diamond q) \;\approx\; \Box\Diamond((\neg p)\mathcal{B}q)$$

The formula on the right states the existence of infinitely many states such that the last observed $p$ was followed (or coincided with) $q$. Because of that, we often refer to the class of recurrence properties also as the *response* class.

Recurrence formulae can specify all the properties specifiable by safety formulae. This is due to the equivalence:

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

They can also specify all the properties specifiable by termination formulae:

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

Examples of properties specifiable by recurrence formulae are accessibility, lack of individual starvation, responsiveness to requests, etc. Recurrence formulae can also express weak-fairness requirements. For simplicity, let us consider a transition $\tau$ with no competitors, i.e., $Comp(\tau) = \{\tau\}$. A weak-fairness requirement for such a transition $\tau$ is that, if $\tau$ is continuously enabled beyond some point, it will eventually be taken. This can be expressed by:

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

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

$$\begin{array}{lcl} (\Box\Diamond\, p \vee \Box\Diamond\, q) & \approx & \Box\Diamond(p \vee q) \\ (\Box\Diamond\, p \wedge \Box\Diamond\, q) & \approx_\omega & \Box\Diamond[p \wedge \circleddash[(\neg p)\mathcal{S} q]] \end{array}$$

- *Persistence*

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

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

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

$$\begin{array}{c} \Box\, p \approx \Diamond\,\Box\,(\boxminus\, p) \\ \Diamond\, p \approx \Diamond\,\Box\,(\diamondminus\, p). \end{array}$$

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

- *Progress*

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

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

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

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

$$\bigwedge_{i=1}^{n}(\Box\Diamond\, p_i \vee \Diamond\,\Box\, q_i)$$

which are identified as *multiple progress* formulae.

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

**Proposition 4.1** *A property $\Pi$, that is specifiable by a temporal formula, is a $\kappa$-property iff it is specifiable by a $\kappa$-formula, where $\kappa \in$ {safety, termination, intermittence, recurrence, persistence, progress}.*

The fact that a $\kappa$-formula specifies a $\kappa$-property is straightforward. The other direction has been proved for the *safety* and *termination* cases. For the other cases we have to rely on a similar proof for automata that we discuss next.

In Figure 2 we present the hierarchy between the temporal characterizations of the properties.

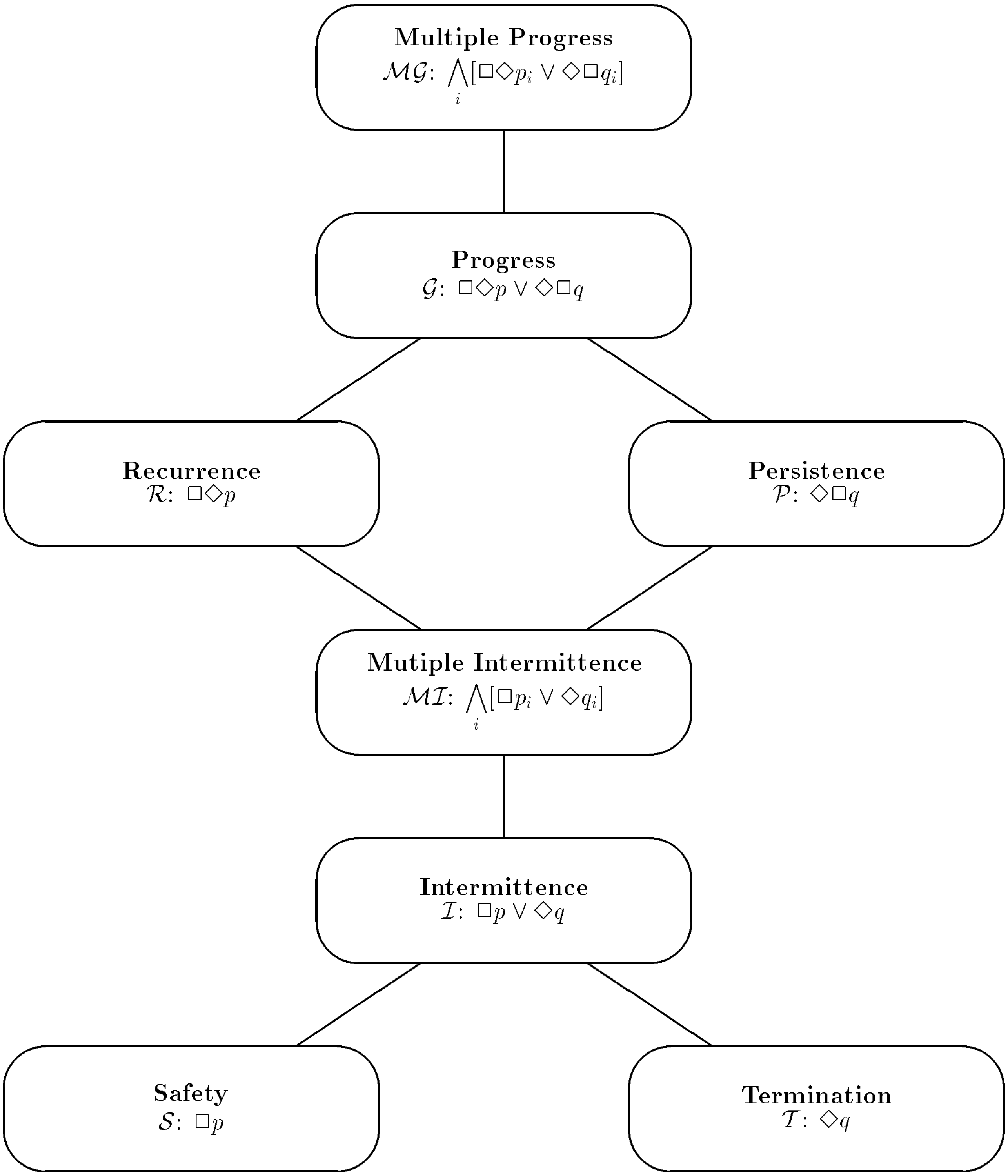



Figure 2: Inclusion Relations between the Temporal Classes

## 4.3 Predicate-Automata

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 $\mathbf{F}$, 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

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

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

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

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 $r$, $\mathit{Inf}(r)$, to be the set of automaton-states that occur infinitely many times in $r$.

A run $r$ is defined to be *accepting* if either $\mathit{Inf}(r) \cap R \neq \phi$ or $\mathit{Inf}(r) \subseteq P$. 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 $q \in Q$,

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

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.

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 *termination* automaton is such that there is no transition from $q \in G$ to $q' \in B$.
- A *intermittence* 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 multiple intermittence as follows:
- A *multiple intermittence* 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 enter from $B$ to $G$ (equivalently, exit from $G$ into $B$), at most $k$ times. It is easy to see that the case of $k = 1$ corresponds to the definition of an intermittence 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 *progress* 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 $\mathit{Inf}(r) \cap R_i \neq \phi$ or $\mathit{Inf}(r) \subset P_i$. 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 multiple progress 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 4.2** *A property* $\Pi$*, that is specifiable by automata, is a* $\kappa$*-property iff it is specifiable by a* $\kappa$*-automaton, where* $\kappa \in$ *{safety, termination, intermittence, recurrence, persistence, progress}.*

For most of these types, this proposition has been proved in [Lan69], with some minor differences in the definitions of a safety and termination automata. The case of progress, 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 an $\kappa$-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

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

Consequently, every property specifiable by a single automaton is a progress 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

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

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

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

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

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

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

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

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

specifying $\Pi$.

Let $\delta: Q \times \Sigma^+ \rightarrow Q$ 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 $\Pi = A_\omega(\mathit{Pref}(\Pi))$.
  We construct an automaton:

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

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

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

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 $\sigma \in \Sigma^+$,

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

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

Assume that $\sigma \notin \mathit{Pref}(\Pi)$. 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 $\sigma' \in \mathit{Pref}(\Pi)$ such that also $\delta(q_0, \sigma') = q$. If $\sigma' \in \mathit{Pref}(\Pi)$, 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 $\mathit{Inf}(r) = \mathit{Inf}(r')$, and hence $\sigma \cdot \sigma''$ should be accepted. This contradicts our assumption that $\sigma \notin \mathit{Pref}(\Pi)$. 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 $\mathit{Pref}(\Pi)$, 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 $\sigma' \notin \mathit{Pref}(\Pi)$. Thus, $\sigma'$ cannot be the prefix of a computation in $\Pi$, and therefore $\sigma \notin \Pi$.

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

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

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 $\Pi = R_\omega(\Pi')$ 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* C in the automaton is a set of states such that there exists a cyclic path in the automaton that passes only through the states in C, and at least once through each of them. We only consider *accessible cycles*. These are cycles such that the path leading from $q_0$ to some $q$ in C, and the cyclic path traversing 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 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 $\sigma^*$, such that $\delta(q_0, \sigma^*) = q$, there exists a positive integer $n$ (possibly dependent on $\sigma^*$) such that $\sigma^* \cdot (\sigma')^n$ contains a prefix $\sigma \prec \sigma^* \cdot (\sigma')^n$, $|\,\sigma| \succ |\sigma^*|$, which belongs to $\Pi'$.

To see this, we observe that the computation $\sigma^* \cdot (\sigma')^\omega$ has $J'$ as infinity set, and is therefore in $\Pi$. Consequently, $\sigma^* \cdot (\sigma')^\omega$ must have infinitely many prefixes in $\Pi'$, most of which are longer than $\sigma^*$. The shortest of these is a prefix of $\sigma^* \cdot (\sigma')^n$ 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 progress properties specifiable by automata is handled as follows.

Let $\Pi$ be a progress 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 progress 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:

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

- 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

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

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 $\mathit{Inf}(r) \supseteq A_i$ 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 $\mathcal{M}'$. ∎

## 4.4 Deciding the Type of a Property

In this section we consider the following problem:

**Problem 4.1** *Given a Streett automaton $\mathcal{M}$, decide whether the property specified by this automaton is a $\kappa$-property, where $\kappa \in$ {safety, termination, intermittence, recurrence, persistence, progress}*

The following proposition gives an answer to this general question:

**Proposition 4.3** *It is decidable whether a given Streett automaton specifies a $\kappa$-property, for $\kappa \in$ {safety, termination, recurrence, persistence, progress}.*

Again, for the first types, the answer has been given by Landweber in [Lan69]. For the case of progress, 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 *termination* property.
  $\mathcal{M}$ specifies a termination 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 *progress* property.
  $\mathcal{M}$ specifies a progress 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 progress 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 $i = 1, \ldots, n$.

## 4.5 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 4.4** *A property 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 progress 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 progress 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 4.5** *A property 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) progress automaton $\langle Q, q_0, T, R, P\rangle$ is

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

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 proposition 4.1.
We illustrate this method on the following part of the proposition.

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

**Proof**

Let $\varphi$ be the formula specifying $\Pi$. Using the translation described in proposition 4.4, we construct a counter-free automaton $\mathcal{M}_\varphi$, specifying the progress property $\Pi$. Using the construction described in proposition 4.2 for the case of a progress property, we construct a progress 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 4.5, to construct a progress formula $\varphi_{\tilde{\mathcal{M}}}$ specifying $\Pi$.

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

# 5 Proof System – General Part

In this section we present a deductive proof system for the general part of temporal logic.

## 5.1 Validity and Provability

We consider two types of validity. A state formula $\varphi$ is defined to be *assertionally valid*, denoted by $\models_A \varphi$, if

$s \models \varphi$ for every state $s \in \Sigma$.

A temporal formula $\varphi$ is defined to be *temporally valid*, denoted by $\models_T \varphi$, if

$(\sigma, 0) \models \varphi$ for every finite or infinite sequence of states $\sigma \in \Sigma^* \cup \Sigma^\omega$.

Note that if the formula $\varphi$ contains some temporal operator, then only one type of validity, the temporal validity can be applied to it. In that case we can simply refer to it as valid, and denote this fact by writing $\models \varphi$. On the other hand, in the case that $\varphi$ is an *assertion* (synonymous term for a state formula), both types of validity apply. Since we are mainly interested in temporal validity, we will drop the subscript $_T$ from the validity notation, and interpret the simpler $\models$ as $\models_T$.

Corresponding to these two types of validity, we may consider two possible deductive proof systems. The first proof system supports proving assertional validity of state formulae, while the second system supports proving (general) temporal validity of temporal formulae. This leads to two notions of *provability*. We say that a state formula $\alpha$ is *assertionally provable*, denoted by $\vdash_A \alpha$, if its assertional validity can be proven using the assertional proof system. Similarly, we say that a formula $\varphi$ is *temporally provable*, denoted by $\vdash_T \varphi$, if its (general) temporal validity can be proven by the temporal proof system we describe in this section. To alleviate the notation, we use $\vdash$ to denote $\vdash_T$ whenever it causes no ambiguity. In most of the presentation of our proof system, we write simply $\varphi$ as a premise, conclusion, or line in a proof, as representing the provability statement $\vdash_T \varphi$.

The notions of validity and provability can be extended to *consequential* validity and provability. Let $\Phi$ denote a set of formulae, $\Phi = \{\varphi_1, ..., \varphi_n\}$, to which we refer as *assumptions*. We say that a formula $p$ is a *valid consequence* of the assumptions $\Phi$, denote by $\Phi \models \varphi$ if every sequence that satisfies all the assumptions in $\Phi$, also satisfies $p$.

We present below the temporal proof system. First we will introduce the future fragment of the operators and their associated proof system, and then extend it by the addition of the past operators.

## 5.2 Future Fragment – Operators and Axioms

We use the boolean operators $\rightarrow, \neg, \vee, \wedge$ and $\equiv$.

As basic temporal (future) operators we take the *Weak Next* operator $\widetilde{\bigcirc}$, and the *Unless* operator $\mathcal{W}$. The semantic definition of these two operators is given by:

- $(\sigma, j) \models \widetilde{\bigcirc} q \iff j+1 = |\sigma|$ or $(\sigma, j+1) \models q$

  Note that the last state in a sequence satisfies $\widetilde{\bigcirc} q$ for any $q$, including $q = \mathrm{F}$.

- $(\sigma, j) \models p \mathcal{W} q \iff \forall k\, (j \le k < |\sigma|) : [(\sigma, k) \models p \ \vee\ \exists i\, (j \le i \le k) : (\sigma, i) \models q]$

  Thus, $p \mathcal{W} q$ holds at position $j$ iff $p$ holds continuously at all positions greater or equal to $j$, either to the end of the sequence $\sigma$, or at least to the first occurrence of $q$.

We can define all the other future operators in terms of these basic two.

Strong Next $\bigcirc q \;=\; \neg\widetilde{\bigcirc}\neg q$

Obviously, $\bigcirc q$ holds at a position in a sequence if there exists a next position, and $q$ holds there.

Henceforth $\square\, p \;=\; (p\,\mathcal{W}\,\mathbf{F})$

Obviously, $\square\, p$ holds at a position $j$ if $p$ holds at all positions $i \geq j$.

Eventually $\diamondsuit\, q \;=\; \neg\,\square\,\neg q$

Obviously, $\diamondsuit\, q$ holds a at a position $j$ if $q$ holds at some position $i \geq j$.

Until $p\,\mathcal{U}\,q \;=\; (p\,\mathcal{W}\,q) \wedge \diamondsuit\, q$

Obviously, $p\,\mathcal{U}\,q$ holds if $p$ holds continuously until the next occurrence of $q$, which is guaranteed to happen.

Entailment $p \Rightarrow q \;=\; \square(p \rightarrow q)$

Clearly, $p$ entails $q$ over a sequence if $p$ implies $q$ at all positions in the sequence.

Strong equivalence $p \Leftrightarrow q \;=\; \square(p \equiv q)$

Clearly, $p$ is strongly equivalent to $q$ over a sequence if $p$ is equivalent to $q$ at all positions in the sequence.

## Axioms

**F1.** $\bigcirc\, p \Rightarrow \widetilde{\bigcirc} p$

**F2.** $\widetilde{\bigcirc}(p \rightarrow q) \;\Rightarrow\; (\widetilde{\bigcirc} p \rightarrow \widetilde{\bigcirc} q)$

**F3.** $\square(p \rightarrow q) \;\Rightarrow\; (\square\, p \rightarrow \square\, q)$

**F4.** $\square\, p \rightarrow \square\, \widetilde{\bigcirc} p$

**F5.** $(p \Rightarrow \widetilde{\bigcirc} p) \rightarrow (p \Rightarrow \square\, p)$

**F6.** $(p\,\mathcal{W}\,q) \;\Leftrightarrow\; [q \vee (p \wedge \widetilde{\bigcirc}(p\,\mathcal{W}\,q))]$

**F7.** $\square\, p \Rightarrow p\,\mathcal{W}\,q$

## Inference Rules

For the first inference rule, we need the notion of the *temporal instantiation* of a state formula. Let $\alpha$ be a state formula containing the proposition symbol $Q$, and $\varphi$ some temporal formula. We denote by $\alpha[\varphi/Q]$ the temporal formula obtained from $\alpha$ by replacing all occurrences of $Q$ by $\varphi$. We call $\alpha[\varphi/Q]$ a temporal instantiation of the state formula $\alpha$ by the replacement $Q \leftarrow \varphi$. For example, the formula

$$[r \wedge (r \rightarrow (p\,\mathcal{U}\,q))] \rightarrow (p\,\mathcal{U}\,q))$$

is a temporal instantiation of the state formula

$$\alpha \; : \; [r \wedge (r \rightarrow Q)] \rightarrow Q,$$

obtained by the replacement $Q \leftarrow (p\mathcal{U}q)$. It is easy to extend this notion to a multiple replacement, where a list of proposition symbols is replaced by a matching list of temporal formulae. We consider only *safe* replacements, which do not cause free variables appearing in the temporal formula $\varphi$ to be captured by quantifications appearing in $\alpha$.

**INS** TEMPORAL INSTANTIATION

For $\varphi$, a temporal instantiation of the state formula $\alpha$,

$$\frac{\models_A \alpha}{\vdash_T \Box\varphi}$$

**MP** MODUS PONENS

$$\frac{p \rightarrow q \;,\; p}{q}$$

**PAR** PARTICULARIZATION

$$\frac{\Box p}{p}$$

The **INS** rule enables us to derive the temporal validity of the formula $[r \wedge (r \rightarrow (p\mathcal{U}q))] \rightarrow (p\mathcal{U}q))$ from the assertional validity of the state formula $[r \wedge (r \rightarrow Q)] \rightarrow Q$. Note that the premise for the rule requires validity of $\alpha$ rather than provability, which makes the temporal proof system independent of the assertional proof system that may be used to derive $\alpha$. In principle, we can use alternative means, such as specific decision procedures, for establishing the assertional validity of $\alpha$.

On the other hand, the price of this generality is that it makes the deductive system non-recursive. This is because the set of assertionally valid statements (allowing quantification and general interpretations into concrete structures) is in general non-recursive. To avoid this situation, we will usually restrict the assertions $\alpha$ to purely propositional formulae. With this restriction, the system becomes recursive. We will maintain this assumption unless we explicitly say otherwise.

Note that, in some sense, the **PAR** rule can be viewed as the inverse of the **INS** rule. While the **INS** rule deduces the validity of $\varphi$ at all states in a sequence from the validity of $\alpha$ at an arbitrary single step, the **PAR** deduces the validity of $p$ at a single step, the first in the sequence, from its validity at all states in the sequence.

## 5.3 Theorems and Derived Rules

In the following, we use the axioms and rules given above to derive some additional theorems and derived rules. A theorem is a statement of the form $\vdash_T p$, claiming that the formula $p$ is provable in the presented deductive system, and hence is valid.

The proof of a theorem is presented as a sequence of lines. A line in a proof can be an instantiation of an axiom, i.e., particularizing symbols such as $p$ and $q$, appearing in the axiom and denoting arbitrary formulae, to more concrete forms, which may in turn contain other symbols standing for arbitrary formulae. For example, having the axiom **F1**: $\bigcirc p \Rightarrow \widetilde{\bigcirc} p$, we may instantiate $p$ to $\square q$ and include $\bigcirc \square q \Rightarrow \widetilde{\bigcirc} \square q$ as a line in a proof, quoting **F1** as justification. Alternately, a line in a proof may be justified by an application of a *rule*. For example, the **MP** rule allows us to infer $q$ from $p \rightarrow q$ and $p$. Using instantiation again, this enables us to write $\widetilde{\bigcirc} r$ at a line $k$ of the proof, provided we have already derived $\square r \rightarrow \widetilde{\bigcirc} r$ at line $i < k$, and $\square r$ at line $j < k$. In this case, we will include **MP** $i, j$ as the justification for line $k$. Once a theorem is established, we are allowed to use it as an axiom in subsequent rules. This is because we can view a theorem as an abbreviation for its proof, which means that in any proof using the theorem as a justification for a line, we could have expanded the derivation of the line by substituting the proof of the theorem at this point.

A derived rule is a statement of the form:

$$\frac{\begin{array}{c} p_1 \\ p_2 \\ . \\ . \\ p_n \end{array}}{q}$$

This statement claims that if the formulae $p_1, ..., p_n$, called *premises*, are provable in the deductive system, then so is the *conclusion* $q$.

The proof of a derived rule is similar to the proof of a theorem in that it includes lines resulting from instantiation of axioms (or previously established theorems), or application of rules. In addition, it may also include lines which list one of the premises $p_i$, with the justification *Premise*. Again, once a rule is proven, we may use it in subsequent proofs to justify additional steps. This is justified again by the claim that any such application at some step in a proof could have been proven without the use of the derived rule, by substituting the proof of the rule (properly instantiated) at that step. In many cases rules are easier to use than their formulation as axioms. Consequently, many of our derivations below will transform axioms into rules.

**Rule PR** PROPOSITIONAL REASONING

$$\frac{\begin{array}{c} (p_1 \wedge p_2 \wedge ... \wedge p_n) \ \rightarrow \ q \\ p_1, p_2, ..., p_n \end{array}}{q}$$

Proof:

| | |
|---|---|
| 1. $(p_1 \wedge p_2 \wedge ... \wedge p_n) \rightarrow q$ | Premise. |
| 2. $p_1, p_2, ..., p_n$ | Premise. |
| 3. $[(p_1 \wedge p_2 \wedge ... \wedge p_n) \rightarrow q] \rightarrow [p_1 \rightarrow (p_2 \rightarrow (...(p_n \rightarrow q)...)]$ | **INS** |
| 4. $p_1 \rightarrow (p_2 \rightarrow (...(p_n \rightarrow q)...)$ | **MP** $3, 1$, def. of $\Rightarrow$. |
| 5. $p_2 \rightarrow (p_3 \rightarrow (...(p_n \rightarrow q)...)$ | **MP** $4, 2(1)$. |
| 6. ... | |
| ... | |
| $q$ | |

■

Often, we use the **PR** rule without explicitly designating the first premise. This is done whenever the first premise is an obvious propositional tautology.

**Rule GMP** GENERALIZED MODUS PONENS

$$\frac{p \Rrightarrow q\ ,\ \Box\, p}{\Box\, q}$$

Proof:

1. $\Box(p \rightarrow q) \Rrightarrow (\Box\, p \rightarrow \Box\, q)$ **F3**
2. $\Box(p \rightarrow q) \rightarrow (\Box\, p \rightarrow \Box\, q)$ **PAR**
3. $p \Rrightarrow q$ Premise
4. $\Box(p \rightarrow q)$ Definition of $\Rrightarrow$
5. $\Box\, p \rightarrow \Box\, q$ **MP** 2, 4
6. $\Box\, p$ Premise
7. $\Box\, q$ **MP** 5, 6

∎

**Rule GPR** GENERALIZED PROPOSITIONAL REASONING

$$\frac{\begin{array}{c}(p_1 \wedge p_2 \wedge ... \wedge p_n) \Rrightarrow q \\ \Box\, p_1, \Box\, p_2, ..., \Box\, p_n\end{array}}{\Box\, q}$$

Proof:

1. $(p_1 \wedge p_2 \wedge ... \wedge p_n) \Rrightarrow q$ Premise.
2. $\Box\, p_1, \Box\, p_2, ..., \Box\, p_n$ Premise.
3. $[(p_1 \wedge p_2 \wedge ... \wedge p_n) \rightarrow q] \Rrightarrow [p_1 \rightarrow (p_2 \rightarrow (...(p_n \rightarrow q)...)]$ **INS**
4. $p_1 \Rrightarrow (p_2 \rightarrow (...(p_n \rightarrow q)...)$ **GMP** $3, 1$, def. of $\Rrightarrow$.
5. $p_2 \Rrightarrow (p_3 \rightarrow (...(p_n \rightarrow q)...)$ **GMP** $4, 2(1)$.
6. ...

   ...

   $\Box\, q$

∎

Often, we use the **GPR** rule without explicitly designating the first premise. This is done whenever the first premise is a substitution instance of an obvious propositional tautology.

**Rule R1**

$$\frac{p \Leftrightarrow q}{p \Rrightarrow q}$$

Proof:

1. $\Box[p \equiv q]$ Premise
2. $[p \equiv q] \Rrightarrow (p \rightarrow q)$ **INS**.
3. $p \Rrightarrow q$ **GMP** $2, 1$ ; def. of $\Rrightarrow$.

∎

**Rule R2**

$$\frac{p \Rrightarrow q\ ,\ q \Rrightarrow p}{p \Leftrightarrow q}$$

Proof:

| | |
|---|---|
| 1. $[(p \rightarrow q) \wedge (q \rightarrow p)] \mathrel{\Rightarrow\!\!\!-} (p \equiv q)$ | **INS**. |
| 2. $\Box(p \rightarrow q)$ | Premise. |
| 3. $\Box(q \rightarrow p)$ | Premise. |
| 4. $p \Leftrightarrow q$ | **GPR**, $1, 2, 3$ |

$\blacksquare$

## 5.4 Theorems and Rules for $\widetilde{\bigcirc}$

**Rule $\widetilde{\bigcirc}$ G** $\widetilde{\bigcirc}$ GENERALIZATION

$$\frac{\Box\, p}{\Box\, \widetilde{\bigcirc} p}$$

Proof:

| | |
|---|---|
| 1. $\Box\, p$ | Premise |
| 2. $\Box\, p \rightarrow \Box\, \widetilde{\bigcirc} p$ | **F4** |
| 3. $\Box\, \widetilde{\bigcirc} p$ | **MP** 2, 1 |

$\blacksquare$

**Rule $\widetilde{\bigcirc}$ M** $\widetilde{\bigcirc}$ IS MONOTONIC

$$\text{(a)} \quad \frac{p \mathrel{\Rightarrow\!\!\!-} q}{\widetilde{\bigcirc} p \mathrel{\Rightarrow\!\!\!-} \widetilde{\bigcirc} q} \qquad \text{(b)} \quad \frac{q \Leftrightarrow p}{\widetilde{\bigcirc} p \Leftrightarrow \widetilde{\bigcirc} q}$$

Proof:

| | | |
|---|---|---|
| (a) | 1. $p \mathrel{\Rightarrow\!\!\!-} q$ | Premise. |
| | 2. $\Box\, \widetilde{\bigcirc}(p \rightarrow q)$ | $\widetilde{\bigcirc}$**G**. |
| | 3. $\widetilde{\bigcirc}(p \rightarrow q) \mathrel{\Rightarrow\!\!\!-} (\widetilde{\bigcirc} p \rightarrow \widetilde{\bigcirc} q)$ | **F2**. |
| | 4. $\widetilde{\bigcirc} p \mathrel{\Rightarrow\!\!\!-} \widetilde{\bigcirc} q$ | **GMP** $3, 2$. |
| (b) | 1. $p \Leftrightarrow q$ | Premise. |
| | 2. $p \mathrel{\Rightarrow\!\!\!-} q$ | **R1**. |
| | 3. $\widetilde{\bigcirc} p \mathrel{\Rightarrow\!\!\!-} \widetilde{\bigcirc} q$ | $\widetilde{\bigcirc}$**M**(a). |
| | 4. $\widetilde{\bigcirc} q \mathrel{\Rightarrow\!\!\!-} \widetilde{\bigcirc} p$ | Similarly. |
| | 5. $\widetilde{\bigcirc} p \Leftrightarrow \widetilde{\bigcirc} q$ | **R2**, $3, 4$. |

$\blacksquare$

**Theorem T1** $\widetilde{\bigcirc}$ IS CONJUNCTIVE

$$\widetilde{\bigcirc}(p \wedge q) \Leftrightarrow (\widetilde{\bigcirc} p \wedge \widetilde{\bigcirc} q)$$

Proof:

| | |
|---|---|
| 1. $p \mathrel{\Rightarrow\!\!\!-} (q \rightarrow (p \wedge q))$ | **INS** |
| 2. $\widetilde{\bigcirc} p \mathrel{\Rightarrow\!\!\!-} \widetilde{\bigcirc}(q \rightarrow (p \wedge q))$ | $\widetilde{\bigcirc}$**M** |
| 3. $\widetilde{\bigcirc}(q \rightarrow (p \wedge q)) \mathrel{\Rightarrow\!\!\!-} (\widetilde{\bigcirc} q \rightarrow \widetilde{\bigcirc}(p \wedge q))$ | **F2** |
| 4. $\widetilde{\bigcirc} p \mathrel{\Rightarrow\!\!\!-} (\widetilde{\bigcirc} q \rightarrow \widetilde{\bigcirc}(p \wedge q))$ | **GPR** $2, 3$ |

| | | |
|---|---|---|
| 5. | $(\widetilde{\bigcirc} p \wedge \widetilde{\bigcirc} q) \Rightarrow \widetilde{\bigcirc}(p \wedge q)$ | **GPR** |
| 6. | $(p \wedge q) \Rightarrow p$ | **INS** |
| 7. | $\widetilde{\bigcirc}(p \wedge q) \Rightarrow \widetilde{\bigcirc} p$ | $\widetilde{\bigcirc}$**M** |
| 8. | $\widetilde{\bigcirc}(p \wedge q) \Rightarrow \widetilde{\bigcirc} q$ | Similarly |
| 9. | $\widetilde{\bigcirc}(p \wedge q) \Rightarrow (\widetilde{\bigcirc} p \wedge \widetilde{\bigcirc} q)$ | **GPR** 7, 8 |
| 10. | $\widetilde{\bigcirc}(p \wedge q) \Leftrightarrow (\widetilde{\bigcirc} p \wedge \widetilde{\bigcirc} q)$ | **R2** 5,9 |

∎

**Theorem T2** $\widetilde{\bigcirc}$ IS DISJUNCTIVE

$$\widetilde{\bigcirc}(p \vee q) \Leftrightarrow (\widetilde{\bigcirc} p \vee \widetilde{\bigcirc} q)$$

Proof:

| | | |
|---|---|---|
| 1. | $\widetilde{\bigcirc}((\neg p) \rightarrow q) \Rightarrow (\widetilde{\bigcirc}(\neg p) \rightarrow \widetilde{\bigcirc} q)$ | **F2** |
| 2. | $\widetilde{\bigcirc}(p \vee q) \Rightarrow (\neg \widetilde{\bigcirc}(\neg p) \vee \widetilde{\bigcirc} q)$ | $\widetilde{\bigcirc}$**M, GPR** |
| 3. | $\widetilde{\bigcirc}(p \vee q) \Rightarrow (\bigcirc p \vee \widetilde{\bigcirc} q)$ | Definition of $\bigcirc$, **GPR** |
| 4. | $\widetilde{\bigcirc}(p \vee q) \Rightarrow (\widetilde{\bigcirc} p \vee \widetilde{\bigcirc} q)$ | **F1, GPR** |
| 5. | $p \Rightarrow (p \vee q)$ | **INS** |
| 6. | $\widetilde{\bigcirc} p \Rightarrow \widetilde{\bigcirc}(p \vee q)$ | $\widetilde{\bigcirc}$**M** |
| 7. | $\widetilde{\bigcirc} q \Rightarrow \widetilde{\bigcirc}(p \vee q)$ | Similarly |
| 8. | $(\widetilde{\bigcirc} p \vee \widetilde{\bigcirc} q) \Rightarrow \widetilde{\bigcirc}(p \vee q)$ | **GPR** 6,7 |
| 9. | $\widetilde{\bigcirc}(p \vee q) \Leftrightarrow (\widetilde{\bigcirc} p \vee \widetilde{\bigcirc} q)$ | **R2** 4, 8 |

∎

**Theorem $\widetilde{\bigcirc}$ D$\bigcirc$** $\widetilde{\bigcirc}$ IS DUAL TO $\bigcirc$

$$\neg \widetilde{\bigcirc} p \Leftrightarrow \bigcirc(\neg p)$$

Proof:

| | | |
|---|---|---|
| 1. | $p \Leftrightarrow \neg(\neg p)$ | **INS** |
| 2. | $\widetilde{\bigcirc} p \Leftrightarrow \widetilde{\bigcirc}(\neg(\neg p))$ | $\widetilde{\bigcirc}$**M** |
| 3. | $\neg \widetilde{\bigcirc} p \Leftrightarrow \neg \widetilde{\bigcirc}(\neg(\neg p))$ | **GPR** |
| 4. | $\neg \widetilde{\bigcirc} p \Leftrightarrow \bigcirc(\neg p))$ | Definition of $\bigcirc$ |

∎

**Theorem T3**

$$\widetilde{\bigcirc}(p \rightarrow q) \Leftrightarrow (\widetilde{\bigcirc} p \rightarrow \widetilde{\bigcirc} q)$$

Proof:

| | | |
|---|---|---|
| 1. | $\widetilde{\bigcirc}(p \rightarrow q) \Rightarrow (\widetilde{\bigcirc} p \rightarrow \widetilde{\bigcirc} q)$ | **F2** |
| 2. | $(\widetilde{\bigcirc} p \rightarrow \widetilde{\bigcirc} q) \Rightarrow (\neg \widetilde{\bigcirc} p \vee \widetilde{\bigcirc} q)$ | **GPR** |
| 3. | $(\widetilde{\bigcirc} p \rightarrow \widetilde{\bigcirc} q) \Rightarrow (\bigcirc(\neg p) \vee \widetilde{\bigcirc} q)$ | **ND, GPR** |
| 4. | $(\widetilde{\bigcirc} p \rightarrow \widetilde{\bigcirc} q) \Rightarrow (\widetilde{\bigcirc}(\neg p) \vee \widetilde{\bigcirc} q)$ | **F1** |
| 5. | $(\widetilde{\bigcirc} p \rightarrow \widetilde{\bigcirc} q) \Rightarrow \widetilde{\bigcirc}((\neg p) \vee q))$ | **T2** |
| 6. | $(\widetilde{\bigcirc} p \rightarrow \widetilde{\bigcirc} q) \Rightarrow \widetilde{\bigcirc}(p \rightarrow q)$ | $\widetilde{\bigcirc}$**M, GPR** |
| 7. | $\widetilde{\bigcirc}(p \rightarrow q) \Leftrightarrow (\widetilde{\bigcirc} p \rightarrow \widetilde{\bigcirc} q)$ | **R2**, 1, 6 |

∎

## 5.5 Theorems and Rules about $\bigcirc$

**Rule $\bigcirc$ M** $\bigcirc$ IS MONOTONIC

$$\text{(a)} \quad \frac{p \Rightarrow q}{\bigcirc p \Rightarrow \bigcirc q} \qquad \text{(b)} \quad \frac{q \Leftrightarrow p}{\bigcirc p \Leftrightarrow \bigcirc q}$$

Proof:

(a)

| | |
|---|---|
| 1. $p \Rightarrow q$ | Premise. |
| 2. $(\neg q) \Rightarrow (\neg p)$ | **GPR** |
| 3. $\widetilde{\bigcirc}(\neg q) \Rightarrow \widetilde{\bigcirc}(\neg p)$ | $\widetilde{\bigcirc}$**M** |
| 4. $\neg\widetilde{\bigcirc}(\neg p) \Rightarrow \neg\widetilde{\bigcirc}(\neg q)$ | **GPR** |
| 5. $\bigcirc p \Rightarrow \bigcirc q$ | Definition of $\bigcirc$ |

(b) From (a) and **R2**.

**Theorem T4** $\bigcirc$ IS CONJUNCTIVE

$$\bigcirc(p \wedge q) \Leftrightarrow (\bigcirc p \wedge \bigcirc q)$$

By **T2** and duality. That is, apply **T2** to $\neg p$ and $\neg q$, and then introduce appropriate negations.

**Theorem T5** $\bigcirc$ IS DISJUNCTIVE

$$\bigcirc(p \vee q) \Leftrightarrow (\bigcirc p \vee \bigcirc q)$$

By **T1** and duality.

**Theorem T6**

$$\bigcirc(p \rightarrow q) \Leftrightarrow (\bigcirc p \rightarrow \bigcirc q)$$

By **T3** and duality.

## 5.6 Theorems about $\square$

**Rule $\square$ M** $\square$ IS MONOTONIC

$$\text{(a)} \quad \frac{p \Rightarrow q}{\square p \Rightarrow \square q} \qquad \text{(b)} \quad \frac{q \Leftrightarrow p}{\square p \Leftrightarrow \square q}$$

Proof:

(a)

| | |
|---|---|
| 1. $p \Rightarrow q$ | Premise. |
| 2. $\square\,\square(p \rightarrow q)$ | Definition of $\Rightarrow$ |
| 3. $\square(p \rightarrow q) \Rightarrow (\square p \rightarrow \square q)$ | **F3**. |
| 4. $\square p \Rightarrow \square q$ | **GMP** 3, 2. |

(b) From (a) and **R2**.

**Rule CI** COMPUTATIONAL INDUCTION

$$\frac{p \Rightarrow \widetilde{\bigcirc} p}{p \Rightarrow \square\, p}$$

Proof:

| | |
|---|---|
| 1. $p \Rightarrow \widetilde{\bigcirc} p$ | Premise. |
| 2. $(p \Rightarrow \widetilde{\bigcirc} p) \rightarrow (p \Rightarrow \square\, p)$ | **F5** |
| 3. $p \Rightarrow \square\, p$ | **MP** 2,1 |

$\blacksquare$

**Theorem T7**

$$\square\, p \Leftrightarrow [p \wedge \widetilde{\bigcirc}(\square\, p)]$$

Proof:

| | |
|---|---|
| 1. $p \mathcal{W} \mathbf{F} \Leftrightarrow \mathbf{F} \vee [p \wedge \widetilde{\bigcirc}(p \mathcal{W} \mathbf{F})]$ | **F6** |
| 2. $p \mathcal{W} \mathbf{F} \Leftrightarrow [p \wedge \widetilde{\bigcirc}(p \mathcal{W} \mathbf{F})]$ | **GPR** |
| 3. $\square\, p \Leftrightarrow [p \wedge \widetilde{\bigcirc}(\square\, p)]$ | Definition of $\square$ |

$\blacksquare$

**Rule $\square$ I** $\square$ INTRODUCTION

$$\frac{q \Rightarrow [p \wedge \widetilde{\bigcirc} q]}{q \Rightarrow \square\, p}$$

Proof:

| | |
|---|---|
| 1. $q \Rightarrow [p \wedge \widetilde{\bigcirc} q]$ | Premise. |
| 2. $q \Rightarrow \widetilde{\bigcirc} q$ | **GPR** |
| 3. $q \Rightarrow \square\, q$ | **CI** |
| 4. $q \Rightarrow p$ | **GPR** 1 |
| 5. $\square\, q \Rightarrow \square\, p$ | $\square$ **M** |
| 6. $q \Rightarrow \square\, p$ | **GPR** 3, 5 |

$\blacksquare$

Theorem **T7** and the $\square$**I** rule identify $\square\, p$ as the *maximal* solution of the following equation for $X$

$$[X \Rightarrow (p \wedge \widetilde{\bigcirc} X)] \tag{2}$$

Theorem **T7** shows that $\square\, p$ satisfies the entailment ( 2) when substituted for $X$. The $\square$**I** rule shows that any other solution $q$ of the entailment must be smaller than $\square\, p$, where the ordering between formulae is defined by

$p \prec q$ iff $p \Rightarrow q$.

**Theorem T8** $\square$ IS REFLEXIVE

$$\square\, p \Rightarrow p$$

Proof: **T7 + GPR** $\blacksquare$

**Theorem T9**

$$\square\, p \Rightarrow \widetilde{\bigcirc}\, \square\, p$$

Proof: **T7 + GPR** $\blacksquare$

**Theorem T10**

$$\square\, p \Rightarrow \widetilde{\bigcirc} p$$

Proof:

1. $\Box\, p \Rightarrow p$ — **T8**
2. $\widetilde{\bigcirc}\,\Box\, p \Rightarrow \widetilde{\bigcirc} p$ — $\widetilde{\bigcirc}$**M**
3. $\Box\, p \Rightarrow \widetilde{\bigcirc}\,\Box\, p$ — **T9**
4. $\Box\, p \Rightarrow \widetilde{\bigcirc} p$ — **GPR** 3,2

$\blacksquare$

**Theorem T11** $\Box$ IS IDEMPOTENT

$$\Box\, p \Leftrightarrow \Box\,\Box\, p$$

Proof:

1. $\Box\,\Box\, p \Rightarrow \Box\, p$ — **T8**
2. $\Box\, p \Rightarrow \widetilde{\bigcirc}\,\Box\, p$ — **T9**
3. $\Box\, p \Rightarrow [\Box\, p \wedge \widetilde{\bigcirc}\,\Box\, p]$ — **GPR**
4. $\Box\, p \Rightarrow \Box\,\Box\, p$ — $\Box$**I**
5. $\Box\, p \Leftrightarrow \Box\,\Box\, p$ — **R2** 1,4

$\blacksquare$

The following theorem generalizes **F4** from implication to entailment.

**Theorem T12**

$$\Box\, p \Rightarrow \Box\,\widetilde{\bigcirc} p$$

Proof:

1. $\Box\, p \Rightarrow \widetilde{\bigcirc} p$ — **T10**
2. $\Box\,\Box\, p \Rightarrow \Box\,\widetilde{\bigcirc} p$ — $\Box$**M**
3. $\Box\, p \Rightarrow \Box\,\Box\, p$ — **T11,GPR**
4. $\Box\, p \Rightarrow \Box\,\widetilde{\bigcirc} p$ — **GPR** 3,2

$\blacksquare$

**Rule $\Box$ G** $\Box$ GENERALIZATION

$$\frac{\Box\, p}{\Box\,\Box\, p}$$

Proof:

1. $\Box\, p$ — Premise.
2. $\Box\, p \Rightarrow \Box\,\Box\, p$ — **T11,R1**
3. $\Box\, p \rightarrow \Box\,\Box\, p$ — **PAR**
4. $\Box\,\Box\, p$ — **MP** 3,1

$\blacksquare$

**Theorem T13** $\Box$ IS CONJUNCTIVE

$$\Box(p \wedge q) \Leftrightarrow (\Box\, p \wedge \Box\, q)$$

Proof:

1. $p \wedge q \Rightarrow p$ — **INS**
2. $\Box(p \wedge q) \Rightarrow \Box\, p$ — $\Box$**M**
3. $\Box(p \wedge q) \Rightarrow \Box\, q$ — Similarly

| | |
|---|---|
| 4. $\Box(p \wedge q) \strictif (\Box p \wedge \Box q)$ | **GPR** 2,3 |
| 5. $p \strictif (q \rightarrow (p \wedge q))$ | **INS** |
| 6. $\Box p \strictif \Box(q \rightarrow (p \wedge q))$ | $\Box$**M** |
| 7. $\Box p \strictif (\Box q \rightarrow \Box(p \wedge q))$ | **T8, GPR** |
| 8. $(\Box p \wedge \Box q) \strictif \Box(p \wedge q)$ | **GPR** |
| 9. $\Box(p \wedge q) \Leftrightarrow (\Box p \wedge \Box q)$ | **R2** 4,8 |

## 5.7 Theorems about $\Diamond$

**Theorem T14** $\Diamond$ IS REFLEXIVE

$$p \strictif \Diamond p$$

Proof:

| | |
|---|---|
| 1. $\Box(\neg p) \strictif (\neg p)$ | **T8** |
| 2. $p \strictif \neg \Box(\neg p)$ | **GPR** |
| 3. $p \strictif \Diamond p$ | Definition of $\Diamond$ |

**Theorem T15** $\Diamond$ IS IDEMPOTENT

$$\Diamond p \Leftrightarrow \Diamond \Diamond p$$

Proof:

| | |
|---|---|
| 1. $\Box(\neg p) \Leftrightarrow \Box \Box(\neg p)$ | **T11** |
| 2. $\neg \Box(\neg p) \Leftrightarrow \neg \Box \Box(\neg p)$ | **GPR** |
| 3. $\neg \Box(\neg p) \Leftrightarrow \neg \Box(\neg\neg \Box(\neg p))$ | $\Box$**M, GPR** |
| 4. $\Diamond p \Leftrightarrow \Diamond \Diamond p$ | Definition of $\Diamond$ |

**Rule $\Diamond$ M** $\Diamond$ IS MONOTONIC

$$\text{(a)} \quad \frac{p \strictif q}{\Diamond p \strictif \Diamond q} \qquad \text{(b)} \quad \frac{q \Leftrightarrow p}{\Diamond p \Leftrightarrow \Diamond q}$$

Proof:

(a)

| | |
|---|---|
| 1. $p \strictif q$ | Premise. |
| 2. $\neg q \strictif \neg p$ | **GPR** |
| 3. $\Box(\neg q) \strictif \Box(\neg p)$ | $\Box$**M** |
| 4. $\neg \Box(\neg p) \strictif \neg \Box(\neg q)$ | **GPR** |
| 5. $\Diamond p \strictif \Diamond q$ | Definition of $\Diamond$ |

(b) From (a) and **R2**.

**Theorem T16** $\Diamond$ IS DISJUNCTIVE

$$\Diamond(p \vee q) \Leftrightarrow (\Diamond p \vee \Diamond q)$$

Proof:

| | |
|---|---|
| 1. $\Box((\neg p) \wedge (\neg q)) \Leftrightarrow (\Box(\neg p) \wedge \Box(\neg q))$ | **T13** |
| 2. $\Box(\neg(p \vee q)) \Leftrightarrow (\Box(\neg p) \wedge \Box(\neg q))$ | $\Box$**M, GPR** |
| 3. $\neg \Box(\neg(p \vee q)) \Leftrightarrow (\neg \Box(\neg p) \vee \neg \Box(\neg q))$ | **GPR** |
| 4. $\Diamond(p \vee q) \Leftrightarrow (\Diamond p \vee \Diamond q)$ | Definition of $\Diamond$ |

**Rule $\Diamond$ T** $\Diamond$ IS TRANSITIVE

$$\frac{p \Rightarrow \Diamond q \ , \ q \Rightarrow \Diamond r}{p \Rightarrow \Diamond r}$$

Proof:

| | |
|---|---|
| 1. $p \Rightarrow \Diamond q$ | Premise. |
| 2. $q \Rightarrow \Diamond r$ | Premise. |
| 3. $\Diamond q \Rightarrow \Diamond \Diamond r$ | $\Diamond$**M** |
| 4. $\Diamond q \Rightarrow \Diamond r$ | **T15, GPR** |
| 5. $p \Rightarrow \Diamond r$ | **GPR** 1,4 |

**Rule $\Diamond$ C** $\Diamond$ IS CONFLUENT

$$\frac{p \Rightarrow \Diamond(q \vee r) \ , \ q \Rightarrow \Diamond s \ , \ r \Rightarrow \Diamond s}{p \Rightarrow \Diamond s}$$

Proof:

| | |
|---|---|
| 1. $q \Rightarrow \Diamond s$ | Premise. |
| 2. $r \Rightarrow \Diamond s$ | Premise. |
| 3. $(q \vee r) \Rightarrow \Diamond s$ | **GPR** 1,2 |
| 4. $\Diamond(q \vee r) \Rightarrow \Diamond \Diamond s$ | $\Diamond$**M** |
| 5. $\Diamond(q \vee r) \Rightarrow \Diamond s$ | **T15** |
| 6. $p \Rightarrow \Diamond(q \vee r)$ | Premise. |
| 7. $p \Rightarrow \Diamond s$ | **GPR** 6,5 |

**Theorem T17**

$$\Diamond p \Leftrightarrow p \vee \bigcirc(\Diamond p)$$

Proof:

| | |
|---|---|
| 1. $\Box \neg p \Leftrightarrow (\neg p) \wedge \widetilde{\bigcirc}(\Box \neg p)$ | **T7** |
| 2. $\neg \Box \neg p \Leftrightarrow p \vee \neg \widetilde{\bigcirc} \neg(\neg \Box \neg p)$ **GPR** | |
| 3. $\Diamond p \Leftrightarrow p \vee \bigcirc(\Diamond p)$ | Definitions of $\Diamond$, $\bigcirc$ |

**Rule $\Diamond$ I** $\Diamond$ INTRODUCTION

$$\frac{p \vee \bigcirc q \Rightarrow q}{\Diamond p \Rightarrow q}$$

Proof: By duality from $\square$ **I**.

Theorem **T17** and the $\Diamond$**I** rule identify $\Diamond p$ as the *minimal* solution of the following equation for $X$

$$[p \vee \bigcirc X \Rightarrow X] \tag{3}$$

Theorem **T17** shows that $\Diamond p$ satisfies the entailment ( 3) when substituted for $X$. The $\Diamond$**I** rule shows that any other solution $q$ of the entailment must be greater or equal to $\Diamond p$.

# 6 General Proof System - The Past

Let us now add the set of *past* operators to the language. Similarly to the future set, they can all be defined in terms of the two basic past operators, the *Weak Previous* operator $\circledsim$, and the *Weak Since* operator $\mathcal{B}$. The semantic definition of these two operators is given by:

- $(\sigma, j) \models \circledsim q \iff j = 0$ or $(\sigma, j-1) \models q$

  Note that the first state in a sequence satisfies $\circledsim q$ for any $q$, including $q = \mathrm{F}$.

- $(\sigma, j) \models p\mathcal{B}q \iff \forall k\,(j \geq k \geq 0) : [(\sigma, k) \models p \;\vee\; \exists i\,(j \geq i \geq k) : (\sigma, i) \models q]$

  Thus, $p\mathcal{B}q$ holds at position $j$ iff $p$ holds continuously at all positions lesser than or equal to $j$, either to the beginning of the sequence $\sigma$, or at least to the first preceding occurrence of $q$.

We can define all the other past operators in terms of these basic two.

Strong Previous $\ominus q \;=\; \neg \circledsim \neg q$

Obviously, $\ominus q$ holds at a position in a sequence if there exists a previous position, and $q$ holds there.

Has Always $\boxminus p \;=\; (p\mathcal{B}\mathrm{F})$

Obviously, $\boxminus p$ holds at a position $j$ if $p$ holds at all positions $i \leq j$.

Once $\diamondminus q \;=\; \neg \boxminus \neg q$

Obviously, $\diamondminus q$ holds a at a position $j$ if $q$ holds at some position $i \leq j$.

Strong Since $p\mathcal{S}q \;=\; (p\mathcal{B}q) \wedge \diamondminus q$

Obviously, $p\mathcal{S}q$ holds if $p$ held continuously since the previous occurrence of $q$, which is guaranteed to have happened.

The following additional axioms complete the proof system, enabling it to also deal with the past operators.

## Axioms

**P1.** $\ominus p \Rightarrow \circledsim p$
**P2.** $\circledsim(p \rightarrow q) \Rightarrow (\circledsim p \rightarrow \circledsim q)$
**P3.** $\boxminus(p \rightarrow q) \Rightarrow (\boxminus p \rightarrow \boxminus q)$
**P4.** $\square p \rightarrow \square \circledsim p$
**P5.** $(p \Rightarrow \circledsim p) \rightarrow (p \Rightarrow \boxminus p)$
**P6.** $(p\mathcal{B}q) \Leftrightarrow [q \vee (p \wedge \circledsim(p\mathcal{B}q))]$
**P7.** $\circledsim \mathrm{F}$

As we can easily see, axioms **P1**–**P6** are almost symmetric to axioms **F1**–**F6** of the future fragment. They deviate from full symmetry in that the principal operator is still the $\Rightarrow$ operator, which is a future operator. However, this is only natural in the anchored mode, where a formula of the form $\boxminus\, p$ is equivalent (because it is evaluated at position 0, which has no past) to $p$. Due to this, there is no sense in writing a formula whose highest temporal operator is a past operator. For the same reason, the two factors of the implication in **P4** start with the future operator $\square$.

The last axiom **P7** is unique to the past, and claims that the first position of each sequence (and this is where we test validity) has no preceding position. This is done by stating that all preceding positions satisfy F. Axiom **P7** is one of the few exceptions ( and in some sense the only necessary one) to the advice we have just given, not to have a past operator at a top level position in the formula. On the other hand, note that we do not have the past analogue of **F7**, which should have been $\boxminus\, p \Rightarrow p \mathcal{B} q$. This is because, using **P7** it can be proven as a theorem, as we do below in **T44**.

In addition to these two sets, each dealing mainly with the operators within its respective fragment, we have two *mixed* axioms, mixing the future with the past.

**F8.** $p \Rightarrow \widetilde{\bigcirc} \ominus p$
**P8.** $p \Rightarrow \widetilde{\ominus} \bigcirc p$

The future axiom **F8** states that if we take one step forward from a $p$-state, we can always take one step backwards, and return to a (actually the same) $p$-state. Axiom **P8** describes a similar situation when the first step is taken backwards.

No additional rules are needed to deal with the past.

## 6.1 Theorems and Derived Rules for the Past

Most of the theorems and derived rules for the past fragment can be obtained by symmetry from the corresponding theorems and rules for the future fragment. We include a proof for the first two rules to illustrate the symmetry elements.

**Rule $\widetilde{\ominus}$ G** $\quad \widetilde{\ominus}$ GENERALIZATION

$$\frac{\square\, p}{\square\, \widetilde{\ominus}\, p}$$

Proof:

1. $\square\, p$ — Premise
2. $\square\, p \rightarrow \square\, \widetilde{\ominus}\, p$ — **P4**
3. $\square\, \widetilde{\ominus}\, p$ — **MP** 2, 1

$\blacksquare$

**Rule $\widetilde{\ominus}$ M** $\quad \widetilde{\ominus}$ IS MONOTONIC

$$\text{(a)} \quad \frac{p \Rightarrow q}{\widetilde{\ominus}\, p \Rightarrow \widetilde{\ominus}\, q} \qquad \text{(b)} \quad \frac{q \Leftrightarrow p}{\widetilde{\ominus}\, p \Leftrightarrow \widetilde{\ominus}\, q}$$

Proof:

(a)
1. $p \Rightarrow q$ — Premise.
2. $\square\, \widetilde{\ominus}(p \rightarrow q)$ — $\widetilde{\ominus}$**G**.
3. $\widetilde{\ominus}(p \rightarrow q) \Rightarrow (\widetilde{\ominus}\, p \rightarrow \widetilde{\ominus}\, q)$ — **P2**.
4. $\widetilde{\ominus}\, p \Rightarrow \widetilde{\ominus}\, q$ — **GMP** 3, 2.

For the rest of the theorems and rules that can be established by symmetry, we list them below without proof.

**Theorem T18** $\text{\textcircled{$\sim$}}$ IS CONJUNCTIVE

$$\text{\textcircled{$\sim$}}(p \wedge q) \Leftrightarrow (\text{\textcircled{$\sim$}}\, p \wedge \text{\textcircled{$\sim$}}\, q)$$

**Theorem T19** $\text{\textcircled{$\sim$}}$ IS DISJUNCTIVE

$$\text{\textcircled{$\sim$}}(p \vee q) \Leftrightarrow (\text{\textcircled{$\sim$}}\, p \vee \text{\textcircled{$\sim$}}\, q)$$

**Theorem $\text{\textcircled{$\sim$}}$ D$\circleddash$** $\text{\textcircled{$\sim$}}$ IS DUAL TO $\circleddash$

$$\neg \text{\textcircled{$\sim$}}\, p \Leftrightarrow \circleddash(\neg p)$$

**Theorem T20**

$$\text{\textcircled{$\sim$}}(p \rightarrow q) \Leftrightarrow (\text{\textcircled{$\sim$}}\, p \rightarrow \text{\textcircled{$\sim$}}\, q)$$

**Rule $\circleddash$ M** $\circleddash$ IS MONOTONIC

$$\text{(a)} \quad \frac{p \strictif q}{\circleddash\, p \strictif \circleddash\, q} \qquad \text{(b)} \quad \frac{q \Leftrightarrow p}{\circleddash\, p \Leftrightarrow \circleddash\, q}$$

**Theorem T21** $\circleddash$ IS CONJUNCTIVE

$$\circleddash(p \wedge q) \Leftrightarrow (\circleddash\, p \wedge \circleddash\, q)$$

**Theorem T22** $\circleddash$ IS DISJUNCTIVE

$$\circleddash(p \vee q) \Leftrightarrow (\circleddash\, p \vee \circleddash\, q)$$

**Theorem T23**

$$\circleddash(p \rightarrow q) \Leftrightarrow (\circleddash\, p \rightarrow \circleddash\, q)$$

## 6.2 Theorems about $\boxminus$

**Rule $\boxminus$ G** $\boxminus$ GENERALIZATION

$$\frac{\Box\, p}{\Box \boxminus p}$$

Proof:

1. $\Box\, p$ — Premise
2. $\Box\, \text{\textcircled{$\sim$}}\, p$ — $\text{\textcircled{$\sim$}}$**G**
3. $\text{\textcircled{$\sim$}}\, p \strictif (p \rightarrow \text{\textcircled{$\sim$}}\, p)$ — **INS**
4. $\Box\, \text{\textcircled{$\sim$}}\, p \strictif \Box(p \rightarrow \text{\textcircled{$\sim$}}\, p)$ — $\Box$**M**
5. $\Box\, \text{\textcircled{$\sim$}}\, p \rightarrow (p \strictif \text{\textcircled{$\sim$}}\, p)$ — **PAR**, Definition of $\strictif$
6. $\Box\, \text{\textcircled{$\sim$}}\, p \rightarrow (p \strictif \boxminus p)$ — **P5, PR**
7. $p \strictif \boxminus p$ — **MP** 6,2
8. $\Box \boxminus p$ — **GMP** 7,1

∎

**Rule $\boxminus$ M** $\boxminus$ IS MONOTONIC

$$\text{(a)} \quad \frac{p \strictif q}{\boxminus p \strictif \boxminus q} \qquad \text{(b)} \quad \frac{q \Leftrightarrow p}{\boxminus p \Leftrightarrow \boxminus q}$$

Proof:

(a)

| | | |
|---|---|---|
| 1. | $p \Rightarrow q$ | Premise. |
| 2. | $\square(p \rightarrow q)$ | Definition of $\Rightarrow$ |
| 3. | $\square \boxminus(p \rightarrow q)$ | $\boxminus$**G** |
| 4. | $\boxminus(p \rightarrow q) \Rightarrow (\boxminus p \rightarrow \boxminus q)$ | **P3** |
| 5. | $\square(\boxminus p \rightarrow \boxminus q)$ | **GMP** 4,3 |
| 6. | $\boxminus p \Rightarrow \boxminus q$ | Definition of $\Rightarrow$ |

(b) From (a) and **R2**.

∎

**Rule RI** REVERSE INDUCTION

$$\frac{p \Rightarrow \tilde{\ominus}\, p}{p \Rightarrow \boxminus p}$$

**Theorem T24**

$$\boxminus p \Leftrightarrow p \wedge \tilde{\ominus} \boxminus p$$

**Rule $\boxminus$ I** $\boxminus$ INTRODUCTION

$$\frac{p \Rightarrow q \wedge \tilde{\ominus}\, p}{p \Rightarrow \boxminus q}$$

Theorem **T24** and the $\boxminus$**I** rule identify $\boxminus p$ as the *maximal* solution of the following equation for $X$

$$[X \Rightarrow (p \wedge \tilde{\ominus} X)] \tag{4}$$

**Theorem T25** $\boxminus$ IS REFLEXIVE

$$\boxminus p \Rightarrow p$$

**Theorem T26**

$$\boxminus p \Rightarrow \tilde{\ominus} \boxminus p$$

**Theorem T27**

$$\boxminus p \Rightarrow \tilde{\ominus}\, p$$

**Theorem T28** $\boxminus$ IS IDEMPOTENT

$$\boxminus p \Leftrightarrow \boxminus \boxminus p$$

**Theorem T29**

$$\boxminus p \Rightarrow \boxminus \tilde{\ominus}\, p$$

**Rule $\boxminus$ PG** $\boxminus$ PAST GENERALIZATION

$$\frac{\boxminus p}{\boxminus \boxminus p}$$

**Theorem T30** $\boxminus$ IS CONJUNCTIVE

$$\boxminus(p \wedge q) \Leftrightarrow (\boxminus p \wedge \boxminus q)$$

## 6.3 Theorems about $\diamondminus$

**Theorem T31** $\diamondminus$ IS REFLEXIVE

$$p \Rightarrow \diamondminus p$$

**Theorem T32** $\diamondminus$ IS IDEMPOTENT

$$\diamondminus p \Leftrightarrow \diamondminus \diamondminus p$$

**Rule $\diamondminus$ M** $\diamondminus$ IS MONOTONIC

$$\text{(a)} \quad \frac{p \Rightarrow q}{\diamondminus p \Rightarrow \diamondminus q} \qquad \text{(b)} \quad \frac{q \Leftrightarrow p}{\diamondminus p \Leftrightarrow \diamondminus q}$$

**Theorem T33** $\diamondminus$ IS DISJUNCTIVE

$$\diamondminus(p \vee q) \Leftrightarrow (\diamondminus p \vee \diamondminus q)$$

**Rule $\diamondminus$ T** $\diamondminus$ IS TRANSITIVE

$$\frac{p \Rightarrow \diamondminus q \ , \ q \Rightarrow \diamondminus r}{p \Rightarrow \diamondminus r}$$

**Rule $\diamondminus$ C** $\diamondminus$ IS CONFLUENT

$$\frac{p \Rightarrow \diamondminus(q \vee r) \ , \ q \Rightarrow \diamondminus s \ , \ r \Rightarrow \diamondminus s}{p \Rightarrow \diamondminus s}$$

**Theorem T34**

$$\diamondminus p \Leftrightarrow p \vee \ominus(\diamondminus p)$$

**Rule $\diamondminus$ I** $\diamondminus$ INTRODUCTION

$$\frac{p \vee \ominus q \Rightarrow q}{\diamondminus p \Rightarrow q}$$

Theorem **T34** and the $\diamondminus$**I** rule identify $\diamondminus p$ as the *minimal* solution of the following equation for $X$

$$[p \vee \ominus X \Rightarrow X] \tag{5}$$

## 6.4 Theorems about $\mathcal{W}$

In this section we consider theorems governing the behavior of the *unless* operator $\mathcal{W}$. First, we establish the monotonicity of $\mathcal{W}$ with respect to both arguments.

**Rule $\mathcal{W}$ I** $\mathcal{W}$ INTRODUCTION

$$\frac{r \Rightarrow [q \vee (p \wedge \widetilde{\bigcirc} r)]}{r \Rightarrow (p \mathcal{W} q)}$$

Proof: Define $\varphi : r \wedge \neg(p \mathcal{W} q)$

| | | |
|---|---|---|
| 1. | $\varphi \Rightarrow [q \vee (p \wedge \widetilde{\bigcirc} r)]$ | Premise |
| 2. | $p \mathcal{W} q \Leftrightarrow q \vee [p \wedge \widetilde{\bigcirc}(p \mathcal{W} q)]$ | **F6** |
| 3. | $\neg(p \mathcal{W} q) \Rightarrow \neg q \wedge [\neg p \vee \bigcirc \neg(p \mathcal{W} q)]$ | **GPR**, definition of $\bigcirc$ |
| 4. | $\varphi \Rightarrow \neg q \wedge [\neg p \vee \widetilde{\bigcirc} \neg(p \mathcal{W} q)]$ | Definition of $\varphi$, **F1, GPR** |
| 5. | $\varphi \Rightarrow p \wedge \widetilde{\bigcirc} r \wedge \widetilde{\bigcirc} \neg(p \mathcal{W} q)$ | **GPR** 1,4 |
| 6. | $\varphi \Rightarrow p \wedge \widetilde{\bigcirc}[r \wedge \neg(p \mathcal{W} q)$ | **T1** |
| 7. | $[r \wedge \neg(p \mathcal{W} q)] \Rightarrow p \wedge \widetilde{\bigcirc}[r \wedge \neg(p \mathcal{W} q)]$ | definition of $\varphi$ |
| 8. | $r \wedge \neg(p \mathcal{W} q) \Rightarrow \square p$ | $\square$ **I** |
| 9. | $r \wedge \neg(p \mathcal{W} q) \Rightarrow p \mathcal{W} q$ | **F7** |
| 10. | $r \wedge \neg(p \mathcal{W} q) \Rightarrow \mathbf{F}$ | **GPR** |
| 11. | $r \Rightarrow p \mathcal{W} q$ | **GPR** |

Axiom **F6** and the $\mathcal{W}$**I** rule identify $p \mathcal{W} q$ as the *maximal* solution of the following equation for $X$

$$X \Rightarrow [q \vee (p \wedge \widetilde{\bigcirc} X)] \tag{6}$$

Axiom **F6** shows that $p \mathcal{W} q$ satisfies the entailment (6) when substituted for $X$. The $\mathcal{W}$**I** rule shows that any other solution $r$ of the entailment must be smaller than $p \mathcal{W} q$.

**Rule $\mathcal{W}$ M** $\mathcal{W}$ IS MONOTONIC

$$\text{(a)} \quad \frac{\begin{array}{c} p \Rightarrow p' \\ q \Rightarrow q' \end{array}}{p \mathcal{W} q \Rightarrow p' \mathcal{W}' q} \qquad \text{(b)} \quad \frac{\begin{array}{c} p \Leftrightarrow p' \\ q \Leftrightarrow q' \end{array}}{p \mathcal{W} q \Leftrightarrow p' \mathcal{W} q'}$$

Proof:

(a)

| | | |
|---|---|---|
| 1. | $p \Rightarrow p'$ | Premise. |
| 2. | $q \Rightarrow q'$ | Premise. |
| 3. | $p \mathcal{W} q \Rightarrow [q \vee (p \wedge \widetilde{\bigcirc}(p \mathcal{W} q))]$ | **F6** |
| 4. | $p \mathcal{W} q \Rightarrow [q' \vee (p' \wedge \widetilde{\bigcirc}(p \mathcal{W} q))]$ | **GPR** 3, 2, 1 |
| 5. | $p \mathcal{W} q \Rightarrow p' \mathcal{W}' q$ | $\mathcal{W}$**I** |

(b) From (a) and **R2**.

**Theorem T35**

$$\square((\neg p) \mathcal{W} p)$$

Proof:

| | | |
|---|---|---|
| 1. | $\square\, \mathbf{T}$ | **INS** |
| 2. | $\square\, \widetilde{\bigcirc} \mathbf{T}$ | $\widetilde{\bigcirc}$**G** |
| 3. | $p \Leftrightarrow (p \wedge \widetilde{\bigcirc} \mathbf{T})$ | **GPR** |
| 4. | $\mathbf{T} \Rightarrow [\neg p \vee (p \wedge \widetilde{\bigcirc} \mathbf{T})]$ | **GPR** |
| 5. | $\square((\neg p) \mathcal{W} p$ | $\mathcal{W}$**I** |

The following theorem shows that $\mathcal{W}$ is conjunctive in its first argument.

**Theorem T36**

$$(p \wedge q) \mathcal{W} r \Leftrightarrow (p \mathcal{W} r) \wedge (q \mathcal{W} r)$$

Proof:

| | | |
|---|---|---|
| 1. | $(p \wedge q) \Rightarrow p$ | **INS** |
| 2. | $(p \wedge q)\mathcal{W}r \Rightarrow p\mathcal{W}r$ | $\mathcal{W}$**M** |
| 3. | $(p \wedge q)\mathcal{W}r \Rightarrow q\mathcal{W}r$ | Similarly |
| 4. | $(p \wedge q)\mathcal{W}r \Rightarrow (p\mathcal{W}r) \wedge (q\mathcal{W}r)$ | **GPR** 2,3 |
| 5. | $p\mathcal{W}r \Rightarrow [r \vee (p \wedge \widetilde{\bigcirc}(p\mathcal{W}r))]$ | **F6** |
| 6. | $q\mathcal{W}r \Rightarrow [r \vee (q \wedge \widetilde{\bigcirc}(q\mathcal{W}r))]$ | **F6** |
| 7. | $(p\mathcal{W}r) \wedge (q\mathcal{W}r) \Rightarrow \{r \vee [(p \wedge q) \wedge \widetilde{\bigcirc}(p\mathcal{W}r \wedge q\mathcal{W}r)]\}$ | **GPR** 5,6 |
| 8. | $(p\mathcal{W}r) \wedge (q\mathcal{W}r) \Rightarrow (p \wedge q)\mathcal{W}r$ | $\mathcal{W}$**I** |
| 9. | $(p \wedge q)\mathcal{W}r \Leftrightarrow (p\mathcal{W}r) \wedge (q\mathcal{W}r)$ | **GPR** 4, 8 |

$\blacksquare$

The following theorem shows that $\mathcal{W}$ is disjunctive in its second argument.

**Theorem T37**

$$p\mathcal{W}(q \vee r) \Leftrightarrow (p\mathcal{W}q) \vee (p\mathcal{W}r)$$

Proof:

| | | |
|---|---|---|
| 1. | $q \Rightarrow (q \vee r)$ | **INS** |
| 2. | $p\mathcal{W}q \Rightarrow p\mathcal{W}(q \vee r)$ | $\mathcal{W}$**M** |
| 3. | $p\mathcal{W}r \Rightarrow p\mathcal{W}(q \vee r)$ | Similarly |
| 4. | $(p\mathcal{W}q \vee p\mathcal{W}r) \Rightarrow p\mathcal{W}(q \vee r)$ | **GPR** 2,3 |
| 5. | $p\mathcal{W}(q \vee r) \Rightarrow \{(q \vee r) \vee [p \wedge \widetilde{\bigcirc}(p\mathcal{W}(q \vee r))]\}$ | **F6** |
| 6. | $p\mathcal{W}q \Leftrightarrow \{q \vee [p \wedge \widetilde{\bigcirc}(p\mathcal{W}q)]\}$ | **F6** |
| 7. | $\neg(p\mathcal{W}q) \Leftrightarrow \{\neg q \wedge [\neg p \vee \neg\widetilde{\bigcirc}(p\mathcal{W}q)]\}$ | **GPR** |
| 8. | $\neg(p\mathcal{W}q) \Rightarrow \{\neg q \wedge [\neg p \vee \bigcirc\neg(p\mathcal{W}q)]\}$ | **GPR,** $\widetilde{\bigcirc}$ **D**$\bigcirc$ |
| 9. | $\neg(p\mathcal{W}q) \Rightarrow \{\neg q \wedge [\neg p \vee \widetilde{\bigcirc}\neg(p\mathcal{W}q)]\}$ | **GPR, F1** |
| 10. | $[p\mathcal{W}(q \vee r) \wedge \neg(p\mathcal{W}q)] \Rightarrow \{r \vee [p \wedge \widetilde{\bigcirc}(p\mathcal{W}(q \vee r)) \wedge \widetilde{\bigcirc}\neg(p\mathcal{W}q)]\}$ | **GPR** 5, 9 |
| 11. | $[p\mathcal{W}(q \vee r) \wedge \neg(p\mathcal{W}q)] \Rightarrow \{r \vee [p \wedge \widetilde{\bigcirc}(p\mathcal{W}(q \vee r)) \wedge \widetilde{\bigcirc}\neg(p\mathcal{W}q))]\}$ | **T1, GPR** |
| 12. | $[p\mathcal{W}(q \vee r) \wedge \neg(p\mathcal{W}q)] \Rightarrow p\mathcal{W}r$ | $\mathcal{W}$**I** |
| 13. | $p\mathcal{W}(q \vee r) \Rightarrow (p\mathcal{W}q \vee p\mathcal{W}r)$ | **GPR** |
| 14. | $p\mathcal{W}(q \vee r) \Leftrightarrow (p\mathcal{W}q) \vee (p\mathcal{W}r)$ | **GPR** 4, 13 |

$\blacksquare$

The following two theorems display some idempotence properties of the $\mathcal{W}$ operator, meaning that if we apply it twice, we get back the original formula. Obviously, since the $\mathcal{W}$ operator has two arguments we can perform a double application in either of them.

**Theorem T38**

$$(p\mathcal{W}q)\mathcal{W}q \Leftrightarrow p\mathcal{W}q$$

Proof:

| | | |
|---|---|---|
| 1. | $(p\mathcal{W}q)\mathcal{W}q \Rightarrow [q \vee (p\mathcal{W}q \wedge \widetilde{\bigcirc}((p\mathcal{W}q)\mathcal{W}q))]$ | **F6** |
| 2. | $(p\mathcal{W}q)\mathcal{W}q \Rightarrow [q \vee p\mathcal{W}q]$ | **GPR** |
| 3. | $q \Rightarrow p\mathcal{W}q$ | **F6, GPR** |
| 4. | $(p\mathcal{W}q)\mathcal{W}q \Rightarrow p\mathcal{W}q$ | **GPR** 2, 3 |
| 5. | $p\mathcal{W}q \Rightarrow (q \vee \widetilde{\bigcirc}(p \wedge \widetilde{\bigcirc}(p\mathcal{W}q)))$ | **F6** |
| 6. | $p\mathcal{W}q \Rightarrow (q \vee \widetilde{\bigcirc}(p\mathcal{W}q \wedge \widetilde{\bigcirc}(p\mathcal{W}q)))$ | **GPR** |
| 7. | $p\mathcal{W}q \Rightarrow (p\mathcal{W}q)\mathcal{W}q$ | **GPR** |
| 8. | $(p\mathcal{W}q)\mathcal{W}q \Leftrightarrow p\mathcal{W}q$ | **GPR** 4, 7 |

$\blacksquare$

**Theorem T39**

$$p\mathcal{W}(p\mathcal{W}q) \Leftrightarrow p\mathcal{W}q$$

Proof:

| | |
|---|---|
| 1. $q \Rightarrow p\mathcal{W}q$ | **F6, GPR** |
| 2. $p\mathcal{W}q \Rightarrow p\mathcal{W}(p\mathcal{W}q)$ | $\mathcal{W}$**M** |
| 3. $p\mathcal{W}(p\mathcal{W}q) \Rightarrow [p\mathcal{W}q \vee [p \vee \widetilde{\bigcirc}(p\mathcal{W}(p\mathcal{W}q))]]$ | **F6** |
| 4. $p\mathcal{W}(p\mathcal{W}q) \Rightarrow \{q \vee [p \wedge \widetilde{\bigcirc}(p\mathcal{W}q)] \vee [p \wedge \widetilde{\bigcirc}(p\mathcal{W}(p\mathcal{W}q))]\}$ | **F6, GPR** |
| 5. $p\mathcal{W}(p\mathcal{W}q) \Rightarrow \{q \vee [p \wedge \widetilde{\bigcirc}(p\mathcal{W}q \vee p\mathcal{W}(p\mathcal{W}q))]\}$ | **T1, GPR** |
| 6. $[p\mathcal{W}q \vee p\mathcal{W}(p\mathcal{W}q)] \Rightarrow p\mathcal{W}(p\mathcal{W}q)$ | **GPR** 2 |
| 7. $p\mathcal{W}(p\mathcal{W}q) \Rightarrow \{q \vee [p \wedge \widetilde{\bigcirc}(p\mathcal{W}(p\mathcal{W}q))]\}$ | $\widetilde{\bigcirc}$**M, GPR** 6, 5 |
| 8. $p\mathcal{W}(p\mathcal{W}q) \Rightarrow p\mathcal{W}q$ | $\mathcal{W}$**I** |
| 9. $p\mathcal{W}(p\mathcal{W}q) \Leftrightarrow p\mathcal{W}q$ | **GPR** 2, 8 |

∎

The following rule establishes transitivity of $\mathcal{W}$.

**Rule $\mathcal{W}$ T** $\mathcal{W}$ IS TRANSITIVE

$$\frac{p \Rightarrow q\mathcal{W}r\ ,\ r \Rightarrow q\mathcal{W}t}{p \Rightarrow q\mathcal{W}t}$$

Proof:

| | |
|---|---|
| 1. $p \Rightarrow q\mathcal{W}r$ | Premise. |
| 2. $r \Rightarrow q\mathcal{W}t$ | Premise. |
| 3. $p \Rightarrow q\mathcal{W}(q\mathcal{W}t)$ | $\mathcal{W}$**M** 1, 2 |
| 4. $p \Rightarrow q\mathcal{W}t$ | **T39** |

∎

We list below several additional theorems concerning the $\mathcal{W}$ operator without proofs. They appear together with their proofs in a form that uses implication instead of entailment in [MP83c]. The changes from the implication proofs to entailment proofs is very uniform, and has been amply demonstrated above.

**Theorem T40**

$$[p\mathcal{W}q \wedge (\neg q)\mathcal{W}r] \Rightarrow p\mathcal{W}r$$

The following two theorems are referred to as *collapsing* theorems, since they collapse a nested unless formula into an unnested one, at the cost of losing some sequencing information.

**Theorem T41**

$$(p\mathcal{W}q)\mathcal{W}r \Rightarrow (p \vee q)\mathcal{W}r$$

**Theorem T42**

$$p\mathcal{W}(q\mathcal{W}r) \Rightarrow (p \vee q)\mathcal{W}r$$

The following theorem establishes in some sense the linearity of time. It states that for every two events, $p$ and $q$, either $p$ weakly precedes $q$, or $q$ weakly precedes $p$. Weak precedence of $p$ before $q$, for example, means that *if* $q$ happens, then $p$ must occur before it or simultaneously with it. If $q$ does not happen, then there is no obligation concerning $p$. This weak precedence is expressed by the formula $(\neg q)\mathcal{W}p$, meaning that $q$ will not happen unless $p$ happened before or together with it.

**Theorem T43**

$$\Box[(\neg p)\mathcal{W}q \vee (\neg q)\mathcal{W}p]$$

## 6.5 Theorems about $\mathcal{B}$

In this section we consider theorems governing the behavior of the *Weak Since* operator $\mathcal{B}$. First, we derive some auxiliary theroems leading to the past analogue of axiom **F7**.

The first theorem states that form any position it is possible to go back to the start point of the sequence.

**Theorem T43** THE START POINT IS REACHABLE

$$\Box \diamondminus \widetilde{\ominus}\, \mathbf{F}$$

Proof:

| | | |
|---|---|---|
| 1. | $\diamondminus \widetilde{\ominus}\, \mathbf{F} \Rightarrow \widetilde{\bigcirc} \ominus \diamondminus \widetilde{\ominus}\, \mathbf{F}$ | **P8** |
| 2. | $\ominus \diamondminus \widetilde{\ominus}\, \mathbf{F} \Rightarrow \diamondminus \widetilde{\ominus}\, \mathbf{F}$ | **T34, GPR** |
| 3. | $\widetilde{\bigcirc} \ominus \diamondminus \widetilde{\ominus}\, \mathbf{F} \Rightarrow \widetilde{\bigcirc} \diamondminus \widetilde{\ominus}\, \mathbf{F}$ | $\widetilde{\bigcirc}$**M** |
| 4. | $\diamondminus \widetilde{\ominus}\, \mathbf{F} \Rightarrow \widetilde{\bigcirc} \diamondminus \widetilde{\ominus}\, \mathbf{F}$ | **GPR** 1,3 |
| 5. | $\diamondminus \widetilde{\ominus}\, \mathbf{F} \Rightarrow \Box \diamondminus \widetilde{\ominus}\, \mathbf{F}$ | **CI** |
| 6. | $\diamondminus \widetilde{\ominus}\, \mathbf{F} \rightarrow \Box \diamondminus \widetilde{\ominus}\, \mathbf{F}$ | **PAR** |
| 7. | $\widetilde{\ominus}\, \mathbf{F} \Rightarrow \diamondminus \widetilde{\ominus}\, \mathbf{F}$ | **T34, GPR** |
| 8. | $\widetilde{\ominus}\, \mathbf{F} \rightarrow \diamondminus \widetilde{\ominus}\, \mathbf{F}$ | **PAR** |
| 9. | $\widetilde{\ominus}\, \mathbf{F}$ | **P7** |
| 10. | $\diamondminus \widetilde{\ominus}\, \mathbf{F}$ | **MP** 8,9 |
| 11. | $\Box \diamondminus \widetilde{\ominus}\, \mathbf{F}$ | **MP** 6,10 |

∎

The next theorem provides the past analogue of **F7**.

**Theorem T44**

$$\boxminus\, p \Rightarrow p\,\mathcal{B}\,q$$

Proof: Define $\varphi : \boxminus\, p \wedge \neg(p\,\mathcal{B}\,q)$

| | | |
|---|---|---|
| 1. | $\varphi \Rightarrow p \wedge \widetilde{\ominus} \boxminus\, p$ | **T24,GPR** |
| 2. | $\varphi \Rightarrow \neg[q \vee (p \wedge \widetilde{\ominus}(p\,\mathcal{B}\,q))]$ | **P6** |
| 3. | $\varphi \Rightarrow [\neg q \wedge (\neg p \vee \ominus \neg(p\,\mathcal{B}\,q))]$ | **GPR** definition of $\bigcirc$ |
| 4. | $\varphi \Rightarrow [p \wedge \widetilde{\ominus} \boxminus\, p \wedge \ominus \neg(p\,\mathcal{B}\,q)]$ | **GPR** 1,3 |
| 5. | $\varphi \Rightarrow [\widetilde{\ominus} \boxminus\, p \wedge \ominus \neg(p\,\mathcal{B}\,q)]$ | **GPR** |
| 6. | $\varphi \Rightarrow [\widetilde{\ominus} \boxminus\, p \wedge \widetilde{\ominus} \neg(p\,\mathcal{B}\,q)]$ | **P1** |
| 7. | $\varphi \Rightarrow \widetilde{\ominus}\, \varphi$ | **T18** definition of $\varphi$ |
| 8. | $\varphi \Rightarrow \boxminus\, \varphi$ | **RI** |
| 9. | $\varphi \Rightarrow \ominus \neg(p\,\mathcal{B}\,q)$ | **GPR** 4 |
| 10. | $\neg(p\,\mathcal{B}\,q) \Rightarrow \mathbf{T}$ | **INS** |
| 11. | $\ominus \neg(p\,\mathcal{B}\,q) \Rightarrow \ominus\, \mathbf{T}$ | $\ominus$**M** |
| 12. | $\varphi \Rightarrow \ominus\, \mathbf{T}$ | **GPR** 9,11 |
| 13. | $\boxminus\, \varphi \Rightarrow \boxminus \ominus\, \mathbf{T}$ | $\boxminus$**M** |
| 14. | $\varphi \Rightarrow \boxminus \ominus\, \mathbf{T}$ | **GPR** 8,13 |
| 15. | $\diamondminus \widetilde{\ominus}\, \mathbf{F} \Rightarrow (\varphi \rightarrow \diamondminus \widetilde{\ominus}\, \mathbf{F})$ | **INS** |
| 16. | $\Box \diamondminus \widetilde{\ominus}\, \mathbf{F}$ | **T43** |
| 17. | $\Box(\varphi \rightarrow \diamondminus \widetilde{\ominus}\, \mathbf{F})$ | **GMP** 15,16 |
| 18. | $\varphi \Rightarrow \diamondminus \widetilde{\ominus}\, \mathbf{F}$ | Definition of $\Rightarrow$ |
| 19. | $\varphi \Rightarrow (\boxminus \ominus\, \mathbf{T} \wedge \diamondminus \widetilde{\ominus}\, \mathbf{F})$ | **GPR** 14,18 |
| 20. | $\varphi \Rightarrow \mathbf{F}$ | **GPR** definitions of $\ominus$, $\diamondminus$ and monotonicity |

| | | |
|---|---|---|
| 21. | $\Box(\varphi \to \mathbf{F})$ | Definition of $\Rightarrow$ |
| 22. | $\Box(\boxminus\, p \wedge \neg(p\mathcal{B}q) \to \mathbf{F})$ | definition of $\varphi$ |
| 23. | $\Box(\boxminus\, p \to (p\mathcal{B}q)$ | **GPR,$\Box$M** |
| 24. | $\boxminus\, p \Rightarrow p\mathcal{B}q$ | Definition of $\Rightarrow$ |

$\blacksquare$

Now that we have brought the past part to a symmetric situation to the future part, we may prove the other $\mathcal{B}$ theorems similarly to the proofs of the corresponding $\mathcal{W}$ theorems.

**Rule $\mathcal{B}$ I** $\mathcal{B}$ INTRODUCTION

$$\frac{r \Rightarrow [q \vee (p \wedge \circleddash r)]}{r \Rightarrow (p\mathcal{B}q)}$$

**Rule $\mathcal{B}$ M** $\mathcal{B}$ IS MONOTONIC

$$\text{(a)} \quad \frac{\begin{array}{c} p \Rightarrow p' \\ q \Rightarrow q' \end{array}}{p\mathcal{B}q \Rightarrow p'\mathcal{B}'q} \qquad \text{(b)} \quad \frac{\begin{array}{c} p \Leftrightarrow p' \\ q \Leftrightarrow q' \end{array}}{p\mathcal{B}q \Leftrightarrow p'\mathcal{B}q'}$$

**Theorem T45**

$$\Box((\neg p)\mathcal{B}p)$$

**Theorem T46**

$$(p \wedge q)\mathcal{B}r \Leftrightarrow (p\mathcal{B}r) \wedge (q\mathcal{B}r)$$

**Theorem T47**

$$p\mathcal{B}(q \vee r) \Leftrightarrow (p\mathcal{B}q) \vee (p\mathcal{B}r)$$

**Theorem T48**

$$(p\mathcal{B}q)\mathcal{B}q \Leftrightarrow p\mathcal{B}q$$

**Theorem T49**

$$p\mathcal{B}(p\mathcal{B}q) \Leftrightarrow p\mathcal{B}q$$

**Rule $\mathcal{B}$ T** $\mathcal{B}$ IS TRANSITIVE

$$\frac{p \Rightarrow q\mathcal{B}r \;,\; r \Rightarrow q\mathcal{B}t}{p \Rightarrow q\mathcal{B}t}$$

**Theorem T50**

$$[p\mathcal{B}q \wedge (\neg q)\mathcal{B}r] \Rightarrow p\mathcal{B}r$$

**Theorem T51**

$$(p\mathcal{B}q)\mathcal{B}r \Rightarrow (p \vee q)\mathcal{B}r$$

**Theorem T52**

$$p\mathcal{B}(q\mathcal{B}r) \Rightarrow (p \vee q)\mathcal{B}r$$

**Theorem T53**

$$\Box[(\neg p)\mathcal{B}q \vee (\neg q)\mathcal{B}p]$$

## 6.6 Completeness and Connections with [LPZ]

In [LPZ85], we presented a proof system for the full temporal logic, including the past operators. Naturally, the version presented there is the *floating* version, for which validity of a formula $p$ is defined by requiring that $p$ holds at *all* positions of all sequences. In order to compare the two approaches, we denote the floating validity by $\models^{fl}$. Denoting the anchored validity used in this paper by $\models$, we observe the following relations holding between the two notions of validity:

$$
\begin{array}{lcl}
\models p & \longleftrightarrow & \models^{fl} (\widetilde{\ominus}\, \mathbf{F} \rightarrow p) \\
\models^{fl} p & \longleftrightarrow & \models \Box\, p
\end{array}
$$

Thus, the two notions can be reduced to one another, and it is only a question of convenience, but not of expressibility, which one we prefer to use.

For completeness, we repeat below the proof system of [LPZ85]. We denote provability according to this proof system by $\vdash^{fl}$.

## Axioms

| | Past Axioms | | Future Axioms |
|---|---|---|---|
| FP1. | $\neg \ominus p \equiv \widetilde{\ominus} \neg p$ | FF1. | $\neg \bigcirc p \equiv \widetilde{\bigcirc} \neg p$ |
| FP2. | $\ominus p \rightarrow \widetilde{\ominus}\, p$ | FF2. | $\bigcirc p \rightarrow \widetilde{\bigcirc} p$ |
| FP3. | $p \rightarrow \widetilde{\ominus} \bigcirc p$ | FF3. | $p \rightarrow \widetilde{\bigcirc} \ominus p$ |
| FP4. | $\widetilde{\ominus}(p \rightarrow q) \rightarrow (\widetilde{\ominus}\, p \rightarrow \widetilde{\ominus}\, q)$ | FF4. | $\widetilde{\bigcirc}(p \rightarrow q) \rightarrow (\widetilde{\bigcirc} p \rightarrow \widetilde{\bigcirc} q)$ |
| FP5. | $\neg \diamondminus\, p \equiv \boxminus \neg p$ | FF5. | $\neg \Diamond\, p \equiv \Box \neg p$ |
| FP6. | $\boxminus(p \rightarrow q) \rightarrow (\boxminus\, p \rightarrow \boxminus\, q)$ | FF6. | $\Box(p \rightarrow q) \rightarrow (\Box\, p \rightarrow \Box\, q)$ |
| FP7. | $\boxminus\, p \rightarrow \widetilde{\ominus}\, p$ | FF7. | $\Box\, p \rightarrow \widetilde{\bigcirc} p$ |
| FP8. | $\boxminus(p \rightarrow \widetilde{\ominus}\, p) \rightarrow (p \rightarrow \boxminus\, p)$ | FF8. | $\Box(p \rightarrow \widetilde{\bigcirc} p) \rightarrow (p \rightarrow \Box\, p)$ |
| FP9. | $p \mathcal{S} q \equiv q \vee [p \wedge \ominus(p \mathcal{S} q)]$ | FF9. | $p \mathcal{U} q \equiv q \vee [p \wedge \bigcirc(p \ominus q)]$ |
| FP10. | $\diamondminus\, \widetilde{\ominus}\, \mathbf{F}$ | FF10. | $p \mathcal{U} q \rightarrow \Diamond\, q$ |

## Inference Rules

R1. For each $p$, a temporal instantiation of a propositional tautology, $\vdash^{fl} p$.

R2. If $\vdash^{fl} p \rightarrow q$ and $\vdash^{fl} p$ then $\vdash^{fl} q$ (Modus Ponence).

R3. If $\vdash^{fl} p$ then both $\vdash^{fl} \boxminus\, p$ and $\vdash^{fl} \Box\, p$.

Following the presentation of the floating proof system, [LPZ85] proves its completeness. This shows that if $\models^{fl} p$ then $\vdash^{fl} p$. We can follow the same ideas, with very little changes, to prove the completeness of the anchored proof system we have presented here.

Instead, we will reduce the completeness problem of the anchored system to that of the floating system. This will also demonstrate the relations between the two.

**Proposition 6.1** *For every temporal formula $p$, if $\vdash^{fl} p$ then $\vdash \Box\, p$.*

the proposition claims that whenever $p$ is provable in the floating proof system, then $\Box\, p$ is provable in the anchored system.

To prove this proposition, it is sufficient to show for each axiom $p$ of the floating system, that $\Box\, p$ is provable in the anchored system, and for each rule $p_1, ..., p_n \vdash^{fl} q$ of the floating system, that the rule $\Box\, p_1, ..., \Box\, p_n \vdash \Box\, q$ is provable in the anchored system. For most of the axioms, we have already proven $\Box\, p$ as one of our theorems above. It is not difficult to prove it for the reamining ones. The situation is similar for the rules. For example, for R2 we have to prove $\Box(p \rightarrow q), \Box\, p \vdash \Box\, q$, which is precisely our **GMP** rule.

Using this proposition and the observations above, we can prove immediately the completeness of the anchored proof system presented in this paper.

**Proposition 6.2 Completeness:** *The anchored proof system presented here is complete for the propositional fragment of temporal logic.*

**Proof:**
Assume that $\models p$. By the observation above, it follows that $\models^{fl} \circleddash \mathrm{F} \rightarrow p$. By the completeness of the floating proof system it follows that $\vdash^{fl} \circleddash \mathrm{F} \rightarrow p$. By the proposition above $\vdash \Box \circleddash \mathrm{F} \rightarrow p$. Using the **PAR** rule, we can infer $\vdash \circleddash(\mathrm{F} \rightarrow p)$. By **P7** and **MP** we can infer $\vdash p$. ▪

# 7 The Program Part

In this section we present the program part of the proof system.

In principle, we should provide a separate proof rule for each of the property classes in our hierarchy. In practice, we concentrate on three particular classes, which have special significance as expressing most of the interesting program properties, and forming a natural generalization of the two classes of invariance and eventuality properties considered in the previous proof systems. These are the classes of:

- *Safety* Properties. These are all the properties which can be expressed by a temporal formula of the form
$$\Box q$$
for some *past* formula $q$.

- *Response (Recurrence)* Properties. These are all the properties which can be expressed by a temporal formula of the form
$$\Box(p \rightarrow \Diamond q), \text{ or alternately, } \Box\Diamond q$$
for some *past* formulae $p$, and $q$. Note that we use the names *response* and *recurrence* synonymously.

- *Progress* Properties. These are all the properties which can be expressed by a temporal formula of the form
$$\Box\Diamond p \rightarrow \Box\Diamond q$$
for some *past* formulae $p$, and $q$.

We provide complete rules for each of these classes. This provides full coverage for the complete temporal logic, since by [LPZ85] (see also [Tho81]), any temporal formula $\varphi$ is equivalent to a conjunction of progress properties. Therefore, to prove the $\mathcal{P}$-validity of $\varphi$, it is sufficient to prove the $\mathcal{P}$-validity of each of the conjuncts, for which we can use the rule for progress properties.

## 7.1 Rules for Safety

In this section we consider proof rules for establishing the $\mathcal{P}$-validity of a safety formula. As we recall, a safety formula has the form $\Box p$ for some past formula $p$. Let us review first the appropriate rule given in [MP83a] for the simpler case that $p$ is a state-formula.

For a transition $\tau$, and state-formulae $p$, and $q$, we define the *verification condition* of $\tau$, relative to $p$ and $q$, to be the implication:
$$\rho_\tau \wedge p \rightarrow q',$$

where $\rho_\tau$ is the transition assertion corresponding to $\tau$, and $q'$, the *primed version* of the assertion $q$, is obtained from $q$ by replacing each variable occurring in $q$ by its primed version. We denote this verification condition by $\{p\}\tau\{q\}$. Since $\rho_\tau$ holds for two states $s$ and $s'$ iff $s'$ is a $\tau$-successor of $s$, and $q'$ states that $q$ holds on $s'$, it is not difficult to see that

> If the verification condition $\{p\}\tau\{q\}$ is valid, then every $\tau$-successor of a $p$-state is a $q$-state.

For a set of transitions $T \subseteq \mathcal{T}$, we denote by $\{p\}T\{q\}$ the verification condition requiring that $\{p\}\tau\{q\}$ holds for each $\tau \in T$. Let us denote by $\mathcal{T}_P$ the set of transitions for the program $P$. Then the following rule is sound and complete for establishing the $P$-validity of the invariance formula $\square\, q$ for a state formula $q$.

**Rule INV** INVARIANCE OF STATE FORMULAE

$$\begin{array}{ll} \text{I1.} & \Theta \rightarrow \varphi \\ \text{I2.} & \varphi \rightarrow q \\ \text{I3.} & \{\varphi\}\mathcal{T}_P\{\varphi\} \\ \hline & \square\, q \end{array}$$

This rule uses an auxiliary assertion $\varphi$ which, by premise I1 holds initially, and by premise I3 is propagated form each state to its successor. This shows that $\varphi$ holds continuously over all computations of $P$. Since, by I2 $\varphi$ implies $q$, it follows that $q$ is also an invariant of the program.

## Generalizing to Past Formulae

Next, we have to extend the **INV** rule to deal with formulae $q$, which are past formula. First, we extend the notion of the primed version of a formula, to apply also to a past formula. Recall that the intended meaning of a primed formula is express the value of a formula in the next state, in terms of the values of the variables in the next state and in terms of values in the current state. This is inductively defined as follows:

- For a state formula $p(\bar{u})$, we define as before

$$(p(\bar{u}))' = p(\bar{u}')$$

- For a *Previous* formula

$$(\ominus\, p)' = p$$

  This corresponds to our intuition that $\ominus\, p$ holds in the next state iff $p$ holds now.

- For a *Since* formula

$$(p\,\mathcal{S}\,q)' = q' \vee ((p\,\mathcal{S}\,q) \wedge p')$$

  This corresponds to the intuition that $p\,\mathcal{S}\,q$ holds in the next state if, either $q$ holds there, or $p\,\mathcal{S}\,q$ holds now and $p$ holds next.

With this definition, we extend the notion of the verification condition $\{p\}\tau\{q\}$ to apply also to past formulae $p$ and $q$, and mean

$$\rho_\tau \wedge p \Rrightarrow q'$$

Note that since we work with temporal formulae, we replaced the previous implication by an entailment, because we expect the implication to hold at *all* positions of the computation, not only at the first one.

With this extension, the general single rule for establishing safety properties is given by
**Rule SAFE** PROVING SAFETY

$$\begin{array}{ll} \text{I1.} & \Theta \wedge \mathbf{first} \rightarrow \varphi \\ \text{I2.} & \varphi \Rightarrow q \\ \text{I3.} & \{\varphi\}\mathcal{T}_P\{\varphi\} \\ \hline & \Box\, q \end{array}$$

In this rule, we replaced the implication in I2 by an entailment. In I1 we also added the conjunct **first** which is an abbreviation for the formula $\widetilde{\circleddash}\,\mathrm{F}$, characterizing the first position in the computation as the only position that has no predecessor. This conjunct is sometimes necessary to ensure that $\varphi$ holds in the first position.

The premises I1 – I3 of the rule are temporal, i.e., each contains at least one temporal operator. How are these to be proven? We remind the reader of the **INS** rule, presented in the general part of the proof system, and recommend it as a general tool for deriving temporal validities from assertional ones. Let $p$ be a state formula containing $Q$ as a propositional symbol, and let $\varphi$ be a temporal formula. We denote by $p[\varphi/Q]$ be the temporal formula obtained from $p$ by replacing all occurrences of $Q$ by $\varphi$. We repeat below the rule.
**Rule INS** TEMPORAL INSTANTIATION

$$\frac{\models_A p}{\Box\, p[\varphi/Q]}$$

Note, in particular, that if $p$ has the form $t \rightarrow r$ then the temporal conclusion is an entailment of the form $t[\varphi/Q] \Rightarrow r[\varphi/Q]$.

**Example 7.1** *Consider the trivial program with a single state variable $x$, the data precondition $x = 0$, and a single transition $\tau$ whose assertion is given by $\rho_\tau : x' = x + 1$. We wish to prove for this program the trivial safety property*

$$\Box((x = 10) \rightarrow \diamondminus(x = 5)).$$

This property claims that any state in which $x = 10$ must have been preceded by a state in which $x = 5$. Note that this trivial property would not be true for a program that advances in steps of 2, rather then steps of 1. To prove this property, we identify $q$ as $(x = 10) \rightarrow \diamondminus(x = 5)$ and intend to use the **SAFE** rule. As the auxiliary formula $\varphi$, we take $(x \geq 5) \rightarrow \diamondminus(x = 5)$. The rule requires showing the following three premises:

I1. $(x = 0) \wedge \mathbf{first} \Rightarrow ((x \geq 5) \rightarrow \diamondminus(x = 5))$

I2. $((x \geq 5) \rightarrow \diamondminus(x = 5)) \Rightarrow ((x = 10) \rightarrow \diamondminus(x = 5))$

I3. $(x' = x + 1) \wedge ((x \geq 5) \rightarrow \diamondminus(x = 5)) \Rightarrow ((x' \geq 5) \rightarrow (\diamondminus(x = 5) \vee (x' = 5)))$

In I3 we have already expanded $(\diamondminus(x = 5))$ into $(\diamondminus(x = 5) \vee (x' = 5))$. All of these apparently temporal formulae can be established by the **INS** rule, using the following three valid state formulae, and their associated instantiations.

S1. $(x = 0) \wedge p \rightarrow ((x \geq 5) \rightarrow r)$

with the replacement of $(\mathbf{first}, \diamondminus(x = 5))$ for the proposition symbols $(p, r)$, respectively.

S2. $((x \geq 5) \rightarrow p) \rightarrow ((x = 10) \rightarrow p)$

with the replacement of $\diamondminus(x = 5)$ for the proposition symbol $p$.

S3. $(x' = x + 1) \wedge ((x \geq 5) \rightarrow p) \rightarrow ((x' \geq 5) \rightarrow (p \vee (x' = 5)))$

with the replacement of $\diamondminus(x = 5)$ for the proposition symbol $p$.

The following theorem establishes that these two rules are adequate for proving all the safety properties.

**Theorem 7.1** *The* **SAFE** *and the* **INS** *rules are complete for proving the $\mathcal{P}$-validity of any safety property, based on pure assertional reasoning.*

The proof of the theorem is based on the construction of a big past invariant which relates the values of variables in an accessible state (i.e., appearing in some computation of $P$) to the boolean values of the temporal sub-formulae of the past formula $q$, whose invariance we wish to establish. Details will be given in the full version of the paper.

## Causality Formulae

Even though, in theory, the completeness theorem above fully settles the question of proving the validity of safety formulae, there is a practical interest in identifying special forms of safety formulae, for which a specific proof methodology exists. One of these subclasses contains the properties expressible by the *causality* formula

$$p \Rightarrow \diamondminus q$$

for state formulae $p$ and $q$, which states that every $p$-state is necessarily preceded by a $q$-state.

To present a proof rule for causality properties, we define first the *inverse verification condition*, denoted by $\{p\}\tau^{-1}\{q\}$ and standing for the entailment

$$\rho_\tau \wedge p' \Rightarrow q$$

The validity of this condition ensures that any $\tau$-predecessor of a $p$-state must be a $q$-state. The condition is extended to sets of transitions $T \subseteq \mathcal{T}$ in the usual way. Then, the following rule is adequate for proving causality properties.

**Rule CAUS** PROVING CAUSALITY PROPERTIES

$$\begin{array}{ll} \text{C1.} & p \rightarrow (\varphi \vee q) \\ \text{C2.} & \Theta \rightarrow \neg\varphi \\ \text{C3.} & \{\varphi\}\mathcal{T}_P^{-1}\{\varphi \vee q\} \\ \hline & p \Rightarrow \diamondminus q \end{array}$$

By premise C1, any state satisfying $p$, either already satisfies $q$, or satisfies the auxiliary state formula $\varphi$. By premise C3, the predecessor of any $\varphi$-state must satisfy $\varphi \vee q$. Thus, if we do not find a $q$ preceding $p$, $\varphi$ propagates all the way to the initial position. However, this contradicts premise C2, according to which the initial position cannot satisfy $\varphi$.

There are several advantages in using this rule, even for proving invariance properties $\square\, p$. To prove such an invariance as a causality property, we reformulate it as $\neg p \Rightarrow \diamondminus \mathrm{F}$, i.e., any instance of a $\neg p$ must be preceded by a contradiction.

## Incremental Proofs

In the previous paragraphs, we have considered how to establish the invariance of some past formulae. Having established some basic invariants of this form, we may want to use them in order to derive more complex properties. For this purpose, we use the generalized Modus Ponens rule **GMP**, introduced in the general part of the proof system, which allows us, once we have established the invariance $\Box\, p$, to use $p$ as an extra assumption.

**Rule GMP** GENERALIZED MODUS PONENS

$$\begin{array}{ll} \text{A1.} & p \Rightarrow q \\ \text{A2.} & \Box\, p \\ \hline & \Box\, q \end{array}$$

The premise A1 states that, having $p$ as a constant assumption, we can prove $q$ at all positions. Premise A2 guarantees that $p$ is indeed an invariant of the program. Then from these two we infer that $q$ holds at all states of the computations of the program $P$.

## 7.2 Rules for Response (Recurrence)

Response properties are those which can be expressed by a formula of the form

$$p \Rightarrow \Diamond\, q, \text{ or equivalently, } \Box(p \rightarrow \Diamond\, q)$$

for some past formulae $p$ and $q$. Now that we have learned, in the previous section, how to generalize rules having assertional premises into rules with temporal premises involving past formulae, it is straightforward to properly adapt the set of rules from [MP83a]. The rules for establishing response properties can be partitioned into *single step* rules and *extended* rules. We consider each group in turn.

## Single Step Response Rules

These are the rules that establish properties which depend on the execution of a single helpful transition (or one of several candidates) to accomplish the guaranteed response $q$. We have three rules in this group, which differ by the type of *fairness* on which they rely.

The first rule is unconditional of any fairness assumption, and only relies on the fact that as long as there are enabled transitions, some transition will eventually be taken.

**Rule B-RESP** BASIC RESPONSE RULE

$$\begin{array}{ll} \text{B1.} & p \Rightarrow (q \vee \varphi) \\ \text{B2.} & \{\varphi\}\mathcal{T}_P\{q\} \\ \text{B3.} & \varphi \Rightarrow En(\mathcal{T}_P) \\ \hline & p \Rightarrow \Diamond\, q \end{array}$$

The rule considers three past formulae $p, q$, and the auxiliary $\varphi$. Premise B1 requires that any $p$ state, either already satisfies $q$, or satisfies $\varphi$. Premise B2 requires that taking any transition from a $\varphi$-state, must lead to a $q$-state. Premise B3 requires that at least one transition must be enabled on each $\varphi$-state. Clearly such a transition must be taken next, resulting in a $q$-state.

The next single step rule relies on weak fairness to ensure that eventually a helpful transition, leading to $q$ will be taken. It assumes a weak fairness requirement $(E, T) \in \mathcal{W}$. We denote by $\overline{T}$ the complementary set of transitions, i.e., the transitions in $\mathcal{T}_P - T$.

**Rule W-RESP** RESPONSE UNDER WEAK FAIRNESS

$$\begin{array}{ll} \text{W1.} & p \Rightarrow (q \vee \varphi) \\ \text{W2.} & \{\varphi\} T \{q\} \\ \text{W3.} & \{\varphi\} \overline{T} \{q \vee \varphi\} \\ \text{W4.} & \varphi \Rightarrow En(E) \\ \hline & p \Rightarrow \Diamond q \end{array}$$

Premise W1 ensures, as before, that $p$ entails $q$ or $\varphi$. Premise W2 states that any transition in $T$ leads from $\varphi$ to $q$. Premise W3 states that any transition not in $T$, either leads from $\varphi$ to $q$, or preserves $\varphi$. Premise W4 ensures that $E$ is enabled as long as $\varphi$ holds. It is not difficult to see that if $p$ happens, but is not followed by a $q$, then $\varphi$ must hold continuously beyond this point, and no transition of $T$ is taken. However, due to W4, this means that $E$ is continuously enabled yet no transition of $T$ is ever taken, which violates the requirement of weak fairness for $(E, T)$.

The last rule relies on a strong fairness requirement $(E, T) \in \mathcal{F}$.

**Rule S-RESP** RESPONSE UNDER STRONG FAIRNESS

$$\begin{array}{ll} \text{S1.} & p \Rightarrow (q \vee \varphi) \\ \text{S2.} & \{\varphi\} T \{q\} \\ \text{S3.} & \{\varphi\} \overline{T} \{q \vee \varphi\} \\ \text{S4.} & \varphi \Rightarrow \Diamond (q \vee En(E)) \\ \hline & p \Rightarrow \Diamond q \end{array}$$

The difference between this rule and its **W**-version is in the fourth premise. While W4 requires that $\varphi$ entails $En(E)$ now, S4 requires the eventual enabling of $E$. Here an occurrence of $p$ not followed by a $q$, leads, as before, to $\varphi$ holding continuously, and no transition of $T$ being taken. However, the weaker premise S4 guarantees that $E$ is enabled infinitely many times, which suffices to violate the strong fairness requirement with respect to $(E, T)$.

## Rules for Extended Response

These rules combine single step response properties to form general response properties, which need more than a single helpful transition for their achievement.

First, we list two basic rules, which can be derived in the general part, and which express the monotonicity and transitivity of the response connective.

**Rule RM** THE RESPONSE CONNECTIVE IS MONOTONIC

$$\begin{array}{l} p \Rightarrow q \\ q \Rightarrow \Diamond r \\ r \Rightarrow t \\ \hline p \Rightarrow \Diamond t \end{array}$$

The next rule represents transitivity

**Rule RT** THE RESPONSE CONNECTIVE IS TRANSITIVE

$$\begin{array}{l} p \Rightarrow \Diamond q \\ q \Rightarrow \Diamond r \\ \hline p \Rightarrow \Diamond r \end{array}$$

The most important rule for establishing extended response properties is based on well founded induction. We assume a *well founded* relation $(\mathcal{A}, \prec)$, and a partial *ranking function* $\delta$, mapping states into the domain $\mathcal{A}$. We denote the fact that $\delta$ is defined by $\delta \in \mathcal{A}$. The following rule uses well founded induction to establish an extended response property.

**Rule WELL** WELL FOUNDED RULE FOR RESPONSE

$$\begin{array}{ll} \text{L1.} & p \Rightarrow (q \vee \varphi) \\ \text{L2.} & \varphi \Rightarrow (\delta \in \mathcal{A}) \\ \text{L3.} & [\varphi \wedge (\delta = \alpha)] \Rightarrow \Diamond[q \vee (\varphi \wedge (\delta \prec \alpha))] \\ \hline & p \Rightarrow \Diamond\, q \end{array}$$

Premise L1 ensures that any state satisfying $p$ must satisfy $q$ or $\varphi$. Premise L2 ensures that $\delta$ is defined as long as $\varphi$ holds. Premise L3 guarantees that if $\varphi$ holds with a certain rank $\alpha$, then eventually we will reach a state, in which either $q$ holds, or $\varphi$ is maintained but with a rank lower than $\alpha$. Since a well founded ranking cannot go on decreasing forever, we must eventually reach a $q$-state.

The adequacy of this set of rules for establishing response properties is summarized in the following theorem.

**Theorem 7.2** *The rules given above are complete for proving the $\mathcal{P}$-validity of any response property, based on pure assertional reasoning.*

## 7.3 Rules for Progress

In this section we deal with *progress* properties, which are the properties that can be expressed by a formula of the form

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

for some past formulae $p$ and $q$. There are several alternative forms in which every progress property can be recast. They are given by

$$\Box\Diamond\, r \rightarrow \Box\Diamond\, q, \text{ and } \Box\Diamond\, r \Rightarrow \Diamond\, q.$$

We prefer to work with an extended form of the last formula

$$p \wedge \Box\Diamond\, r \Rightarrow \Diamond\, q$$

This formula states that any occurrence of $p$, which is followed by infinitely many occurrences of $r$, must eventually be followed by a $q$.

### Progress under Weak Fairness

If we work only under the assumption of weak fairness, that is, the family of strong fairness requirements happens to be empty, then we can base the proof of progress properties on some response properties and a well founded argument. This is given by the **W-PROG** rule.

**Rule W-PROG** PROGRESS UNDER WEAK FAIRNESS

$$\begin{array}{ll} \text{W1.} & p \Rightarrow (q \vee (\varphi \wedge (\delta \in \mathcal{A}))) \\ \text{W2.} & [\varphi \wedge (\delta = \alpha)] \Rightarrow [(\varphi \wedge (\delta \preceq \alpha))\, \mathcal{W} q] \\ \text{W3.} & [r \wedge \varphi \wedge (\delta = \alpha)] \Rightarrow \Diamond[q \vee (\delta \prec \alpha)] \\ \hline & p \wedge \Box\Diamond\, r \Rightarrow \Diamond\, q \end{array}$$

This rule uses the $\mathcal{W}$ operator (*Unless*).

Premise W1 of the rule ensures that any position that satisfies $p$, either already satisfies $q$, or satisfies $\varphi$ and has a defined rank. In the later case it initiates a gradual process evolving towards the eventual realization of $q$. Premise W2 ensures that, starting at a position satisfying $\varphi$ and having a defined rank $\alpha$, $\varphi$ is continuously maintained and the rank never increases above $\alpha$ until $q$ occurs, if ever. Premise W3 indicates that an additional occurrence of $r$, strengthens the non-increase, guaranteed by W2, into a decrease. Thus, if there are infinitely many occurrences of $r$ then, either $\delta$ decreases infinitely often, which is impossible due to well foundedness, or $q$ is eventually realized.

The adequacy of this rule is stated by the following proposition

**Proposition 7.1** *For a program with no strong fairness requirements, the* **W-PROG** *rule is complete for proving the $\mathcal{P}$-validity of any progress property.*

Obviously, a progress property $p \wedge \Box\Diamond\, r \Rightarrow\!\!\!\succ \Diamond\, q$ can be valid over a program due to the fact that the simpler response property $p \Rightarrow\!\!\!\succ \Diamond\, q$ is valid. The proposition above essentially identifies the main mechanism, by which the occurrence of infinitely many $r$'s can help a progress property to be realized. This mechanism is associated with a ranking function, which measures the distance away from the realization of $q$, such that each additional occurrence of $r$ causes an eventual decrease in the rank.

## Progress under Strong Fairness

When we have strong fairness requirements, a well founded decrease is not the only mechanism by which infinitely many occurrences of $r$ can cause the program to progress from $p$ to $q$. Another candidate mechanism is that we have a strong fairness requirement $(E, T) \in \mathcal{F}$, such that each transition in $T$ leads from $p$ to $q$, and each occurrence of $r$ causes $E$ to eventually become enabled (at least once). Consequently, the **W-PROG** is no longer adequate.

To cover the case of strong fairness, we present first a single step rule for progress under strong fairness. The rule concerns a strong fairness requirement $(E, T) \in \mathcal{F}$, and past formulae $p, r, q$, and $\varphi$.

**Rule S-PROG** PROGRESS UNDER STRONG FAIRNESS

$$
\begin{array}{ll}
\text{S1.} & p \Rightarrow\!\!\!\succ (q \vee \varphi) \\
\text{S2.} & \{\varphi\} T \{q\} \\
\text{S3.} & \{\varphi\} \overline{T} \{q \vee \varphi\} \\
\text{S4.} & [\varphi \wedge \Box\Diamond(\varphi \wedge r)] \Rightarrow\!\!\!\succ \Diamond(q \vee En(E)) \\
\hline
 & [p \wedge \Box\Diamond\, r] \Rightarrow\!\!\!\succ \Diamond\, q
\end{array}
$$

This rule establishes one step progress, under the assumption of strong fairness for $(E, T)$. This and other progress properties can now be combined, using several properties of the progress connective, such as monotonicity and transitivity. These are summarized below.

**Rule PM** THE PROGRESS CONNECTIVE IS MONOTONIC

$$
\begin{array}{l}
p' \Rightarrow\!\!\!\succ p \\
r' \Rightarrow\!\!\!\succ r \\
q \Rightarrow\!\!\!\succ q' \\
[p \wedge \Box\Diamond\, r] \Rightarrow\!\!\!\succ \Diamond\, q \\
\hline
[p' \wedge \Box\Diamond\, r'] \Rightarrow\!\!\!\succ \Diamond\, q'
\end{array}
$$

The next rule represents transitivity

**Rule PT** THE PROGRESS CONNECTIVE IS TRANSITIVE

$$\frac{\begin{array}{l}[p \wedge \Box\Diamond r] \Rightarrow\!\!\succ \Diamond q \\ [q \wedge \Box\Diamond r] \Rightarrow\!\!\succ \Diamond t\end{array}}{[p \wedge \Box\Diamond r] \Rightarrow\!\!\succ \Diamond t}$$

Finally, we have a well founded rule for combining together progress properties into an unbounded chain.

**Rule WELL-PROG** WELL FOUNDED RULE FOR PROGRESS

$$\frac{\begin{array}{ll}\text{L1.} & p \Rightarrow\!\!\succ (q \vee \varphi) \\ \text{L2.} & \varphi \Rightarrow\!\!\succ (\delta \in \mathcal{A}) \\ \text{L3.} & [\varphi \wedge (\delta = \alpha) \wedge \Box\Diamond r] \Rightarrow\!\!\succ \Diamond[q \vee (\varphi \wedge (\delta \prec \alpha))]\end{array}}{[p \wedge \Box\Diamond r] \Rightarrow\!\!\succ \Diamond q}$$

We summarize this more general case by the following proposition.

**Proposition 7.2** *The rules given above are complete for proving the $\mathcal{P}$-validity of any progress property, based on pure assertional reasoning.*


## Acknowledgement

We gratefully acknowledge the help rendered by Orna Lichtenstein and Ascher Wilk in the preparation of this manuscript. Special thanks are due to Roni Rosner for his dedicated technical help and most helpful suggestions.


# References


[AL88] M. Abadi and L. Lamport, The existence of refinement mappings, *Proc. 3rd IEEE Symp. Logic in Comp. Sci.*, 1988, pp. 165–175.

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

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

[AS89] B. Alpern and F.B. Schneider, Verifying temporal properties without temporal logic, *ACM Trans. Prog. Lang. Syst.* **11**, 1989.

[Bac80] R.J.R. Back, *Correctness preserving program refinements: Proof theory and applications*, Mathematical Center Tracts, 131, Center for Mathematics and Computer Science (CWI), Amsterdam, 1980.

[BK83] R.J.R Back and R. Kurki-Suonio, Decentralization of process nets with a centalized control, *Proc. 2nd ACM Symp. Princ. of Dist. Comp.*, 1983, pp. 131–142.

[BKP84] H. Barringer, R. Kuiper, and A Pnueli, Now you may compose temporal logic specifications, *Proc. 16th ACM Symp. Theory of Comp.*, 1984, pp. 51–63.

[BMP83] M. Ben-Ari, Z. Manna, and A Pnueli, The temporal logic of branching time, *Acta Informatica* **20**, 1983, pp. 207–226.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla, Automatic verification of finite state concurrent systems using temporal logic specifications, *ACM Trans. Prog. Lang. Syst.* **8**, 1986, pp. 244–263.

[CM88] K.M. Chandy and J Misra, *Parallel Program Design*, Addison-Wesley, 1988.

[Dij76] E.W. Dijkstra, *A Discipline of Programming*, Prentice-Hall, New Jersey, 1976.

[dRZ87] W.P. de Roever and J. Zwiers, *Different Styles of Compositional and Modular Proofsystems for a CCS/CSP-like Language*, Technical Report, Philips Research, 1987.

[EH86] E.A. Emerson and J.Y. Halpern, 'Sometimes' and 'not never' revisited: On branching time versus linear time, *J. ACM* **33**, 1986, pp. 151–178.

[GPSS80] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi, On the temporal analysis of fairness, *Proc. 6th ACM Symp. Princ. of Prog. Lang.*, 1980, pp. 163–173.

[Gri81] D. Gries, *The Science of Programming*, Springer, New-York, 1981.

[Jon87] B. Jonsson, *Compositional Verification of Distributed Systems*, Ph.D. thesis, Uppsala University, Sweden, 1987.

[Kam68] J.A.W. Kamp, *Tense Logic and the Theory of Order*, Ph.D. thesis, UCLA, 1968.

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

[KdR83] R. Koymans and W.P. de Roever, Examples of a real-time temporal logic specifications, *The Analysis of Concurrent Systems*, Springer, 1983, pp. 231–252.

[KP87] S. Katz and D. Peled, *Interleaving Set Temporal Logic*, Technical Report 456, Dept. of Computer Science, Technion, Haifa, Israel, 1987.

[KVdR83] R. Koymans, J. Vytopyl, and W.P. de Roever, Real-time programming and asynchronous message passing, *Proc. 2nd ACM Symp. Princ. of Dist. Comp.*, 1983.

[Lam77] L. Lamport, Proving the correctness of multiprocess programs, *Trans. on Software Engineering* **1**, 1977.

[Lam83a] L. Lamport, Specifying concurrent program modules, *ACM Trans. Prog. Lang. Syst.* **5**, 1983, pp. 190–222.

[Lam83b] L. Lamport, What good is temporal logic, *Proc. IFIP Congress* (R.E.A. Mason, ed.), North-Holland, 1983, pp. 657–668.

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

[LP84] O. Lichtenstien and A. Pnueli, Checking that finite state concurrent programs satisfy their linear specification, *Proc. 10th ACM Symp. Princ. of Prog. Lang.*, 1984, pp. 97–107.

[LPZ85] O. Lichtenstien, 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.

[MP83a] Z. Manna and A. Pnueli, How to cook a temporal proof system for your pet language, *Proc. 9th ACM Symp. Princ. of Prog. Lang.*, 1983, pp. 141–154.

[MP83b] Z. Manna and A. Pnueli, Proving precedence properties: the temporal way, *Proc. 10th Int. Colloq. Aut. Lang. Prog.*, Lec. Notes in Comp. Sci. 154, Springer, 1983, pp. 491–512.

[MP83c] Z. Manna and A. Pnueli, Verification of concurrent programs: A temporal proof system, *Foundations of Computer Science IV, Distributed Systems: Part 2* (J.W. DeBakker and J. Van Leuwen, eds.), Mathematical Centre Tracts 159, Center for Mathematics and Computer Science (CWI), Amsterdam, 1983, pp. 163–255.

[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 ∀-automata, *Proc. 14th ACM Symp. Princ. of Prog. Lang.*, 1987, pp. 1–12.

[MW78] Z. Manna and R. Waldinger, Is 'sometime' sometimes better than 'always'?: intermittent assertions in proving program correctness, *Comm. ACM* **21**, 1978, pp. 159–172.

[NGO85] V. Nguyen, D. Gries, and S. Owicki, A model and temporal proof system for network of processes, *Proc. 12th ACM Symp. Princ. of Prog. Lang.*, 1985, pp. 121–131.

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

[Pnu86] A. Pnueli, Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends, *Current Trends in Concurrency*, Lec. Notes in Comp. Sci. 224, Springer, 1986, pp. 510–584.

[QS83] J.P. Queille and J. Sifakis, Fairness and related properties in transition systems — A temporal logic to deal with fairness, *Acta Informatica* **19**, 1983, pp. 195–220.

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

[Rei85] W. Reisig, *Petri Nets: An Introduction*, Volume 4 of *EATCS Monographs on Theoretical Computer Science*, Springer, 1985.

[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. Street, Propositional dynamic logic with converse, *Information and Control* **54**, 1982, pp. 121–141.

[Tho81] W. Thomas, A combinatorial approach to the theory of $\omega$-automata, *Inf. and Cont.* **48**, 1981, pp. 261–283.

[VW86] M.Y. Vardi and P. Wolper, Automata theoretic techniques for modal logics of programs, *J. Comp. Sys. Sci.* **32**, 1986, pp. 183–221.

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

[WGS87] J. Widom, D. Gries, and F.B. Schneider, Completeness and incompleteness of trace-based network proof system, *Proc. 14th ACM Symp. Princ. of Prog. Lang.*, 1987, pp. 27–38.

[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.

[Zwi88] J. Zwiers, *Compositionality, Concurrency and Partial Correctness: Proof theories for networks of processes and their connection*, Ph.D. thesis, University of Eindhoven, The Netherlands, 1988. To appear in the LNCS-series, Springer.