# Proving Precedence Properties: The Temporal Way

by

Zohar Manna and Amir Pnueli

## Department of Computer Science

Stanford University
Stanford, CA 94305

# PROVING PRECEDENCE PROPERTIES:

# THE TEMPORAL WAY

ZOHAR MANNA
Computer Science Department
Stanford University
Stanford, CA
*and*
Applied Mathematics Department
The Weizmann Institute of Science
Rehovot, Israel

AMIR PNUELI
Applied Mathematics Department
The Weizrnann Institute of science
Rehovot, Israel

Abstract:

The paper explores the three important classes of temporal properties of concurrent programs: invariance, liveness and precedence. It presents the first methodological approach to the precedence properties, while providing a review of the invariance and liveness properties. The approach is based on the *unless* operator $\mathcal{U}$, which is a weak version of the *until* operator $\mathcal{U}$. For each class of properties, we present a single complete proof principle. Finally, we show that the properties of each class are decidable over finite state programs.

## 1. INTRODUCTION

In studying temporal properties of programs, i.e., properties that go beyond partial correctness, an obvious hierarchy of such properties can be developed. One way of classifying the different sets in this hierarchy is by the syntax of the temporal formulas expressing them.

The first set in this hierarchy is the class of *invariance* properties *(safety* in the terminology of [L1]). These are the properties that can be expressed in terms of a formula of the form:

$$\Box \psi \quad \text{or} \quad \varphi \supset \Box \psi.$$

A formula of the first form, stated for a program $P$, says that every computation of $P$ continuously satisfies $\psi$. In the case of the second form, the formula says that, whenever $\varphi$ is true, $\psi$ is immediately realized and will hold continuously throughout the rest of the computation. Properties

falling into this class include partial correctness, clean behavior (error freedom), mutual exclusion, and deadlock absence.

The second set in the hierarchy of properties is the class of *Ziveness* properties *(eventualities* in the terminology of [MP1]). These are properties that are expressible by temporal formulas of the form:

$$\diamond \psi \quad \text{or} \quad \varphi \supset \diamond \psi.$$

In both forms these formulas guarantee the occurrence of some event $\psi$, in the first case unconditionally and in the second case conditionally on an earlier occurrence of the event $\varphi$. Among the properties falling into this class are: total correctness, termination, accessibility, lack of individual starvation, and responsiveness.

While most of the researchers in the field tend to agree that these two classes are the first two rungs in a natural hierarchy, there is less of a consensus about what should be the next step in the hierarchy. In previous work we have proposed that the next class to be studied is that of *precedence* properties. In a broad sense, precedence properties are all the properties that are expressible using the *until* operator $\mathcal{U}$. To remind the reader, the expression $p\mathcal{U}q$, read "$p$ until $q$", means that eventually $q$ must happen and between now and then $p$ must continuously hold.

A more mathematical formulation of this definition is given by:

Let $a = s_0, s_1, s_2, \ldots$ be a sequence of states, then $p\mathcal{U}q$ is true for $\sigma$ if there exists a $j \geq 0$ such that:

$q$ is true for the sequence $s_j, s_{j+1}, s_{j+2}, \ldots$

(if $q$ is a state property then $q$ holds at $s_j$), and for every $i, 0 \leq i < j$:

$p$ is true for the sequence $s_i, s_{i+1}, s_{i+2}, \ldots$

(if p is a state property then $p$ holds at $s_i$). Here, a state property is a property that depends only on the state and not on the full sequence. Note that in the special case that $j = 0$, then $q$ is true on $\sigma$ and no requirements for $p$ are implied.

A derived operator is the *precede* operator $\mathsf{P}$ that can be defined by:

$$p\mathsf{P}q \quad \equiv \quad \sim\big((\sim p)\mathcal{U}q\big).$$

The meaning of this operator is that "$p$ precede $q$", i.e., if $q$ ever happens it cannot happen unless $p$ occurs first (strictly before $q$). In contrast to $p\mathcal{U}q$ which requires that $q$ eventually happens, $p\mathsf{P}q$ is automatically satisfied if $q$ never happens.

We often use nested *until* expressions of the form

$$p_1\mathcal{U}\big(p_2\mathcal{U}\big(p_3\mathcal{U}\ldots(p_k\mathcal{U}q)\ldots\big)\big),$$

where $p_1$, ● $,p_{kd}$ are state properties, i.e., formulas dependent, only on the state and containing no temporal operators. By careful examination of the semantic definition of the until operator

we arrive at the interpretation that, stated at to, this expression means that there exist instants $t_1, \ldots, t_k$,

$$to \leq t_1 \leq t_2 \leq \ldots \leq t_k,$$

such that:

$p_1$ holds in every $t$, to $\leq t < t_1$

$p_2$ holds in every $t, t_1 \leq t < t_2$

$p_k$ holds in every $t, t_{k-1} \leq t < t_k$, and

$q$ holds in $t_k$.

Thus, this expression predicts a period of continuous $p_1$ followed by a period of continuous $p_2$, and so on, until a period of continuous $p_k$, followed by an occurrence of $q$. Note that any of these periods may be empty by having $t_i = t_{i+1}$ for an empty $(i+1)$st period.

Since we are interested only in nested until expressions where the nesting is in the second argument, we can omit the parentheses and represent the expression above by:

$$p_1 \, \mathcal{U} \, p_2 \, \mathcal{U} \, p_3 \ldots p_k \, \mathcal{U} \, q.$$

The class of precedence properties that we consider are therefore formulas of one of the forms:

$p \supset (q \mathcal{P} r)$ — a *precede* formula

$p \supset (p_1 \mathcal{U} p_2 \mathcal{U} \ldots p_k \mathcal{U} q)$ — an *until* formula.

Several interesting properties fall into the broad class of precedence properties.

Example:

Let us consider a program G (granter) serving as an allocator of a single resource between several processes (requesters) $R_1, \ldots, R_k$ competing for the resource. Let each $R_i$ communicate with G by means of two boolean variables: $r_i$ and $g_i$. The variable $r_i$ is set to *true* by the requester $R_i$ to signal a request for the resource. Once $R_i$ has the resource it signals its release by setting $r_i$ to *false*. The allocator G signals $R_i$ that the resource is granted to him by setting $g_i$ to true. Having obtained a release signal from $R_i$, which is indicated by $r_i = false$, some time later, it will reappropriate the resource by setting $g_i$ to *false.*

Several obvious and important properties of this system belong to the invariance and liveness classes. For instance, the property

$$\square \ \ \textstyle\sum_{i=1}^{k}(g_i) \leq 1,$$

ensuring that the resource is granted to at most one requester at a time, is an invariant property. In summing boolean variables we treat *true* as 1 and *false* as 0. Similarly, the important property

$$r_i \supset \diamond g_i,$$

3

which ensures *responsiveness,* is a liveness property. It guarantees that every request $r_i$ will eventually be granted by setting $g_i$ to *true*.

Let us, however, consider some precedence properties which are relevant to the specification of such a system.

*(a) Absence of Unsolicited Response.*

An important but often overlooked desired feature is that the resource will not be granted to a party who has not requested it. (A similar property in the context of a communication network is that every message received must have been sent by somebody.) This is expressible by the temporal formula

$$\sim g_i \supset (r_i \mathcal{P} g_i).$$

The formula states that if presently $g_i$ is false, i.e., $R_i$ does not presently have the resource, then before the resource will be granted to $R_i$ the next time, $R_i$ must signal a request by setting $r_i$ to *true.*

*(b) Strict $(FIFO)$ Responsiveness.*

Sometimes the weak commitment of eventually responding to a request is not sufficient. At the other extreme we may insist that responses are ordered in a sequence paralleling the order of arrival of Lhe corresponding requests. Thus if requester $R_i$ succeeded in placing his request before . requester $R_j$ the grant to $R_i$ should precede the grant to $R_j$. A straightforward translation of this sentence yields the following intuitive but slightly imprecise expression:

$$(r_i \mathcal{P} r_j) \supset (g_i \mathcal{P} g_j).$$

A more precise expression which also better conforms to the general form of the class of properties we discuss in this paper is:

$$(r_i \wedge \sim r_j \wedge \sim g_j) \supset (\sim g_j \mathcal{U} g_i).$$

It states that if we ever find ourselves in a situation where $r_i$ is presently on, and $r_j$ and $g_j$ are both off, then we are guaranteed to eventually get a $g_i$, and until that moment, no grant will be made to $R_j$. Note that $r_i \wedge \sim r_j$ implies that $R_i$'s request preceded $R_j$'s request, which has not materialized yet. We implicitly rely here on the assumption that once a request has been made it is not-withdrawn until the request has been honored.

This assumption can also be made explicit as part of the specification, using another precedence expressio n:

$$r_i \supset g_i \mathcal{P}(\sim r_i).$$

Note that while all the earlier properties are requirements from the granter, and should be viewed as the "post-condition" part of the specification, this requirement is the responsibility of the requesters. It can be viewed as part of the "pre-condition" of the specification. Without this assumption, we could not hope to implement the granter in any reasonable way, since it would have to respond to very short and inteimittent requests.

4

*(c) Bounded Over taking.*

The requirement of $FIFO$ responsiveness may sometimes be too restrictive and difficult to implement. Any program for the allocator that scans the requests in a certain polling order, $r_1, \ldots, r_k$ and then back to $r_1$ may respond to requests in, say, the order of their *detection* by the program. This order may be different, from the arrival order. A more realistic requirement would allow deviations from the $FIFO$ discipline, provided they are bounded. For example 1-bounded overtaking would say that for every $i$ and j such that $r_i$ preceded $r_j$, we may allow $g_j$ to precede $g_i$ at most once. $FIFO$ responsiveness may then be regarded as 0-bounded overtaking. In order to express k-bounded overtaking we have to use nested until expressions.

The 1-overtaking property can be expressed by a nested *until* expression:

$$\left(r_i \wedge \sim r_j\right) \supset \left(\sim g_j\right) \mathcal{U} g_j \mathcal{U} \left(\sim g_j\right) \mathcal{U} g_i.$$

This expression predicts a period in which $R_j$ does not have the resource, followed by a continuous period in which $R_j$ has got the resource, followed by a period in which $R_j$ does not have the resource, followed by a grant of tho resource to $R_i$. Since any of these periods may be empty, the formula actually states that in the *worst* case, $R_j$ may gain the resource at most once before $R_i$. ∎

Proofs of invariance properties for concurrent programs, have been extensively discussed in the literature (e.g., [OG], [K],[L1],[MP2]). Fewer suggestions have been made for approaches to proving liveness properties (e.g., [OL],[MP2],[MP3]).

In this work we address the problem of verifying properties of the precedence class. Our main conclusion is that the verification of precedence properties does not call for radically new ideas and can actually be viewed as a generalization of the approaches suggested for invariance and liveness properties. In fact, precede formulas are in many respects generalization of invariance properties, whereas *until* formulas can be established by a generalization of the proof principles for liveness properties.

To provide a proper framework, we first introduce an abstract operational model of concurrent programs. We then outline a proof system based on temporal logic; the system has been shown in [MP5] to be relatively complete for proving all properties of concurrent programs. We then discuss some derived proof principles that are tailored directly for the verification of precedence properties. The utility of these principles is demonstrated by proving several examples.

## 2. A COMPUTATIONAL MODEL

We start by defining an abstract computational model; the temporal logic properties will be stated and proven for computations over this model.

The abstract model consists of the following elements:

S — A set, of *computation states*. This is a possibly infinite set. Every element $s \in S$ represents the full configuration of the computing system; for concrete programs each state includes the values of all the program variables as well as the program pointers for all the processes.

5

$\theta$ -- The *initiality predicate.* We will only consider computations originating in a state $s_0$ such that $\theta(s_0)$ holds.

$T$ ----- A finite set *of transitions.* With each transition $\tau \in T$ we associate a partial function $f_\tau : S \rightarrow 2^S$, where $f_\tau(s)$ yields all the possible outcomes of the transition $\tau$ on the state $s \in S$. A transition $\tau \in T$ is said to be *enabled* on a state $s$ if $f_\tau(s) \neq \phi$; otherwise it is called *disabled* on $s$. A state s such that no transition $\tau \in T$ is enabled on it is called *terminal.*

$J$ — The *justice family.* This is a (possibly empty) family of sets of transitions $J = \{T_1^J, \ldots, T_k^J\}$. Each set in J, $T_i^J \subset T$, is called a *justice set* and a justice requirement defined below is to be applid to the set $T_i^J$.

$3$ — The *fairness family.* This is a (possibly empty) family of sets of transitions $\mathcal{F} = \{T_1^F, \ldots T_\ell^F\}$. Each set in 3, $T_j^F \subset T$, is called a *fairness set* and a fairness requirement is to be applied to $T_j^F$.

An *initialized computation* of such a system is a sequence of states with labelled transitions:

$$\sigma : \quad s_0 \xrightarrow{\tau_1} s_1 \xrightarrow{\tau_2} s_2 \xrightarrow{\tau_3} \ldots \quad \text{where } \tau_i \in T,$$

which satisfies the following requirements:

- *Maximality.* The sequence $\sigma$ is maximal, i.e., either it is infinite or the last state $s_k$ is terminal.

- *Initiality.* The first state $s_0$ satisfies the initiality predicate, i.e., $\theta(s_0) = true$.

- *State- to-State transition.* For each step $s_i \xrightarrow{\tau_{i+1}} s_{i+1}$ in $\sigma$ we have that $s_{i+1} \in f_{\tau_{i+1}}(s_i)$.

- *Justice.* For each $T^J \in J$ we impose *a justice requirement:*

  - $\sigma$ is finite, or

  - $\sigma$ is infinite and contains an infinite number of states on which no transition in $T^J$ is enabled, or

  - an infinite number of a-steps are labelled by transitions in $T^J$.

  This corresponds to the notion that if for all states from a certain point on, some transition in $T^J$ (not necessarily always the same) is always enabled, then some transition of $T^J$ will be taken infinitely many times.

- *Fairness.* For each $T^F \in \mathcal{F}$ we impose a fairness *requirement:*

  - $\sigma$ 'is finite, or

  - $\sigma$ is infinite and from a certain point on no transition of $T^F$ is enabled, or

  - some transition of $T^F$ is taken infinitely many times.

  This corresponds to the notion that if some transitions from $T^F$ are enabled infinitely many times then some transitions from $T^F$ are activated infinitely many times.

6

An *admissible computation* is any suffix of an initialized computation.

When considering a concrete computational system, we have to identify the five elements described above with more concrete objects. Since our example is based on a shared-variables computational model, we proceed with such identification for the *shared-variables system*. Such a system has the form:

$$\overline{y} := g(\overline{x}); \ [P_1 \parallel \ \ldots \ \parallel P_m],$$

where $\overline{y} = (y_1, \ldots, y_n)$ are the program (shared) variables, $\overline{x} = (x_1, \ldots, x_\ell)$ are the input variables, and $P_1, \ldots, P_m$ are the concurrent processes of the program. Each $P_i$ is represented by a transition graph with nodes (locations) $L_i = (\ell_0^i, \ldots, \ell_t^i)$ and directed edges $E_i = \{e_1^i, \ .\ \ldots\ e_r^i\}$. The locations $\ell_0^i$ are the *entry* locations of $P_i$, respectively. Each edge $e \in E_i$ is labelled by an instruction:
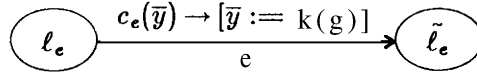


whose meaning is that when $c_e(\overline{y})$ is true, execution may proceed from $\ell_e$ to $\tilde{\ell}_e$ while assigning the values $h_e(\overline{y})$ to the variables $\overline{y}$. Special cases are the semaphore instructions request(y) and *release(y)*, equivalent to $(y > 0) \to [y := y - 1]$ and $true \to [y := y + 1]$, respectively. We refer the reader to [MP1] for a more detailed discussion of these models.

A program state for this system has the form:

$$\langle \ell^1, \ldots, \ell^m; \ \eta_1, \ldots, \eta_n \rangle,$$

where each $\ell^i \in L_i$ denotes the current location of the execution in the process $P_i$, and each $\eta_j \in D$ is the current value of the program variable $y_j$. (The variables $\overline{y}$ are assumed to range over some domain $D$.) Thus we identify the set of all states S as the set of all $(m + n)$-tuples $(L_1 \ \cdot \ \ldots \ x \ L_m \ x \ D^n)$.

The initiality predicate is given by:

$$0(\ell^1, \ldots, \ell^m; \overline{y}) : \quad \left[ \bigwedge_{i=1}^{m} (\ell^i = \ell_0^i) \right] \ \wedge \ (\overline{y} = g(\overline{x}))$$

ensuring that all the processes are at their initial locations and the values of the program variables are properly initialized.

The set of transitions $T$ is identified with the set of all edges $\bigcup_{i=1}^{m} E_i$. For $\tau = e \in E_i$ w e 'define

$$(\ell^1, \ldots, \ell^m; \ \tilde{\overline{\eta}}) \in f_\tau(\ell^1, \ldots \ \ell^m; \overline{\eta})$$

if and only if

$$\ell^i = \ell_e, \quad \tilde{\ell}^i = \tilde{\ell}_e, \quad \tilde{\ell}^j = \ell^j \ \text{for every} \ j \neq i, \quad c_e(\overline{\eta}) = true \quad \text{a n d} \ \tilde{\overline{\eta}} = h_e(\overline{\eta}).$$

The justice family is given by:

$$J \ = \ \{E_1, \ldots, \qquad E_m\};$$

that is, we require that justice be applied to each process individually. This implies that in any infinite computation, each process that has not terminated yet will eventually be scheduled.

The fairness family is given by:

$$\mathcal{I} = \{\{e\} \mid e \text{ is labelled by a } request(y) \text{ instruction}\}.$$

Thus, each semaphore transition is to be individually treated fairly. This implies that a *request(y)* instruction which is waiting while y turns positive infinitely many times must eventually be performed.

In considering computations of a program as models for temporal formulas that express properties of the program, we define the model $\tilde{\sigma}$ corresponding to a sequence $\sigma$,

$$\sigma: \quad s_0 \xrightarrow{\tau_1} s_1 \xrightarrow{\tau_2} s_2 \xrightarrow{\tau_3} \ldots,$$

as follows: If $\sigma$ is infinite then the corresponding model is

$$\tilde{\sigma}: \quad s_0, s_{1}, {}_3 \ {}_2 \ , \ldots.$$

In the case that $\sigma$ is finite and its last state is the terminal state $s_k$, we take $\tilde{\sigma}$ to be

$$\tilde{\sigma}: \quad s_0, s_1, \ldots, s_k, s_k, \ldots,$$

that is, the last state repeats forever.


## 3. THE PROOF SYSTEM

The proof system consists of three parts.

- Part A, called the *general part,* formalizes the pure temporal logic properties of sequences in general. It is completely independent of the particular program analyzed.

- Part B, called the *domain-dependent* part, formalizes the properties of the domain over which the program operates, such as integers, reals, strings, lists, trees, etc.

- Part C is the *program-dependent* part. It provides a formalization of the properties that result from restricting our attention to the computational sequences of the particular program being analyzed.

We refer the reader to [MP4], [MP5] for a discussion of parts A and B. Here we only repeat part C which we further develop in order to prove precedence properties.

The program-dependent part consists of four axiom schemes corresponding to the four requirements imposed on admissible computations. In the following, a *state formula* is a formula containing no temporal operators and hence interpretable on a single state.

Let $\varphi$ and $\psi$ be two state formulas. We say that a transition $\tau$ *leads from* $\varphi$ *to* $\psi$ if for every two states $s$ and $s'$ the following is true:

$$\varphi(s) \wedge \left(s' \in f_\tau(s)\right) \;\Rightarrow\; \psi(s').$$

Note that this formula is classical, i.e., contains no temporal operators and should be expressible and provable in the first-order theory over the domain.

For example, in the case of the shared-variables computation model a transition $\tau$ would correspond to an edge e in some process $P_i$:

$$\ell^i \xrightarrow[\ e\ ]{\ c(g) \to [\bar{y} := h(\bar{y})]\ } \tilde{\ell}^i$$
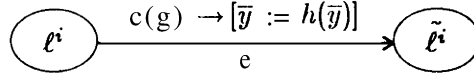
so that the condition above is expressible as

$$\varphi(\ell^1, \ldots, \ell^i, \ldots, \ell^m; \bar{y}) \wedge c(\bar{y}) \quad \Rightarrow \quad \psi(\ell^1, \ldots) \tilde{\ell}^i, \ldots, \ell^m; h(y)).$$

Given a subset of transitions $T' \subset T$, we say that $T'$ leads from $\varphi$ to $\psi$ if every transition $\tau \in T''$ leads from $\varphi$ to $\psi$. If the full set $T$ leads from $\varphi$ to $\psi$, we also say that the *program P leads from $\varphi$ to $\psi$.*

The state formula *Terminal,* characterizes the terminal states:

$$Terminal(s) = \bigwedge_{\tau \in T} (f_\tau(s) = \phi).$$

Also, for a subset $T'$ of transitions, the state formula *Enabled* characterizes the enabled transitions in $T'$:

$$Enabled(T')(s) = \bigvee_{\tau \in T'} [f_\tau(s) \neq \phi].$$

Both formulas are expressible by a quantifier-free first-order formula.

Following are the inference rules of the program part:

| |
|---|
| **(INIT)**   For an arbitrary temporal formula w |
| $\vdash \quad \theta \supset \Box w$ |
| $\vdash \ \Box \ \bullet$ |

This rule states that if w is an invariant for all initialized computations it is also an invariant for all admissible computations. This is because every admissible computation is a suffix of an initialized computation, and a property of the form Cl w is hereditary -from a sequence to all of its suffixes.

| |
|---|
| **(TRNS)** Let $\varphi$ and $\psi$ be two state formulas |
| $\vdash$   Every $\tau \in T$ leads from $\varphi$ to $\psi$ |
| $\vdash (\varphi \wedge Terminal) \supset \psi$ |
| $\vdash \ \varphi \supset \bigcirc \psi$ |

The first premise ensures that as long as at least one transition is enabled, then if the current state satisfies $\varphi$, the next state must satisfy $\psi$. The second premise handles the case that all

9

transitions are disabled, i.e., that of a terminal state. In a computation this means that no further action is possible and the next state is identical to the present. Hence this premise also ensures that in such a case the next state will satisfy $\psi$.

---

(JUST)    Let $\varphi$ and $\psi$ be two state formulas, and $T^J \in J$ a justice set

  ⊢  Every $\tau \in T$ leads from $\varphi$ to $\varphi \vee \psi$

  ⊢  Every $\tau \in T^J$ leads-from $\varphi$ to $\psi$

---

    ⊢  $[\varphi \quad \Box \quad |Enabled(TJ)] \supset \varphi \, \mathcal{U} \, \psi$

---

To justify this rule, consider a computation $\sigma$ such that $\varphi \wedge \Box \, Enabled(T^J)$ holds for $\sigma$ but $\varphi \, \mathcal{U} \, \psi$ does not hold. By the first premise, once $\varphi$ holds it can only stop holding when $\psi$ happens. Hence $\varphi \, \mathcal{U} \, \psi$ may fail to hold only if $\psi$ never happens and $\varphi$ is true forever. Since we assumed that $T^J$ is continuously enabled on $\sigma$, some transition in $T^J$ must eventually be activated, and this in a state satisfying $\varphi$. Hence, by the second premise, once this transition is activated, it achieves $\psi$, contrary to our assumption.

A similar rule applies to fairness:

---

(FAIR)    Let $\varphi$ and $\psi$ be two state formulas, and $T^F \in \mathfrak{J}$ a fairness set

  ⊢  Every $\tau \in T$ leads from $\varphi$ to $\varphi \vee \psi$

  ⊢  Every $\tau \in T^F$ leads from $\varphi$ to $\psi$

---

    ⊢  $[\varphi \wedge \quad \Box \, | \Diamond Enabled(T^F)] \supset \varphi \, \mathcal{U} \, \psi$

---

The justification is similar to that of the JUST rule.

In the following discussion we will consider computations only under the assumption of justice. This amounts to considering an empty fairness family $\mathfrak{J} = \phi$. In the shared-variables computation system this means that we consider programs without semaphores. The reintroduction of fairness to the following analysis can be done in a straightforward manner.

In [MP5] the set of the rules above has been shown to be relatively complete. By this we mean that an arbitrary property which is valid for a given program, can be proved using these rules, provided the pure logic and domain dependent parts are strong enough to prove all valid properties. This result implies that the program dependent part is adequate for establishing all the properties that are true for admissible computations. However, while giving full generality, these rules do not provide specific guidance for proving properties of the three important classes that we have discussed: invariance, liveness and precedence.

We will proceed to develop derived rules, one for each class. These rules, while being derivable in the general system, have the advantage of being complete for their classes. By this we mean, that every valid property in the class can be proved using a single application of the proposed rule as the only temporal step. All the premises to the rule are first-order over the domain. Thus, for anyone who is interested only in proving properties of these classes, the respective rules are the only temporal proof rules he may ever need, dispensing for example with the general temporal logic part.

10

Wc will illustrate these rules on a single example -- an algorithm for mutual exclusion (Fig. 0) -- taken from [Pe]. The program consists of two concurrent proccsscs, $P_1$ and $P_2$ that compete on the access to Lheir critical regions, presented by $\ell_3$ and $m_3$ rcspcctively. Entry into the critical regions is expected to bc exclusive, i.e., at no time can $P_1$ be at $\ell_3$ while at the samc time $P_2$ is at $m_3$. The processes communicate by mcans of the shared-variables $y_1, y_2, t$. Process $P_i$ sets $y_i$ ($i = 1, 2$) to $T$ whenever he is interested in entering his critical region. IIe then procccds to set $t$ to $i$. Following, he reaches a waiting stats ($\ell_2$ or $m_2$, respectively). There he waits until either $y_{\bar{i}} = F$ (here $\bar{i}$ is the compcting process, i.c., $\bar{1} = 2$ and $\bar{2} = 1$) or $t = \bar{i}$. In the first case he infers that the competitor is not currently interested. In the sccond case hc infers that $P_{\bar{i}}$ is interested but has arrived to his waiting state *after* $P_i$ did, since $P_{\bar{i}}$ *was* the *last* to set $t$ to $\bar{i}$. In any of these cases $P_i$ enters his critical region. Once he finishcs his business there hc exits while setting $y_i$ to $F$, indicating loss of interest in further cntrics for the present.

This description is of course intuitive and informal. The following discussions will provide more formal proofs of the correctness of the algorithm.

## 4. INVARIANCE PROPERTIES

A single rulc which is complete for this class is:

(INV) — Invariance Rule

Let $\varphi$ and $\psi$ be slate properties

A . $\vdash \theta \supset \varphi$

B. I- Every $\tau \in T$ leads from $\varphi$ to $\varphi$

c . $\vdash \varphi \supset \psi$

———————————————————

$\vdash \Box \psi$

A slightly more elaborate rule can similarly be used to establish properties of the forrn $\varphi \supset \Box \psi$.

Since the rule is derivable from the INIT and TRNS rules above, it is certainly sound.

To argue that it is complete for properties of the form $\Box \psi$, let $\psi$ bc a state property such that $\Box \psi$ is true for all computations. Define the predicate:

$$Acc(s) = \{\text{There exists an initialized computation segmcnt } s_0 \xrightarrow{\tau_1} s_1 \xrightarrow{\tau_2} \ldots \xrightarrow{\tau_k} s_k = s\}.$$

Thus, $Acc(s)$ is truc for a state $s$ iff there cxisls an initialized computation having $s$ as one of its slates. Wc have dcfined $Acc(s)$ in words rathcr than by a formula; howcvcr, if Lhe underlying domain is rich cnough to contain, say, the intcgcrs, then this predicate is expressible by a first-order formula over the domain.

We now apply the INV rule with $\varphi = Acc$. Certainly $\theta \supset Acc$, since every stale so satisfying $\theta$ participates in a computaLion: $s_0 \to s_1 \to \ldots$. It is also easy to sce that if $s$ is accessible and $s' \in f_\tau(s)$ then s' is also accessible. This cstablishcs premisc B. Premise C says that every accessiblc state satisfics $\psi$, but this follows from our assumption that $\Box \psi$ is true on all admissible . compuLations. Conscqucntly the INV rulc is always applicable.

Let us consider some invariance properties for the mutual exclusiion program (Fig. 0) presented above. $I_0 : \quad \vdash \square \quad ((t = 1) \vee (t = 2))$

Note that for this prograrn

$$\theta : \quad at\ell_0 \wedge at m_0 \wedge [(y_1, y_2, t) = (F, F, 1)].$$

T a k e $\varphi = \psi = (t = 1) \vee (t = 2)$. It is easy to verify that $\theta \supset \varphi$ since $\theta$ implies $t = 1$. Similarly by inspecting every transition we see that all of them maintain $\varphi$.

$I_1 : \quad \vdash \square(y_1 \equiv \ell_{1..3})$

The proposition $\ell_{1..3}$ is defined as $at\ell_1 \vee at\ell_2 \vee at\ell_3$, i.e., it holds whenever $P_1$ is somewhere in $\{\ell_1, \ell_2, \ell_3\}$. Potentially falsifying transitions are:

$\ell_0 \to \ell_1$: setting both $y_1$ and $\ell_{1..3}$ to $T$.

$\ell_3 \to \ell_0$: setting both $y_1$ and $\ell_{1..3}$ to F.

All other transitions do not rnodify either $y_1$ or $\ell_{1..3}$.

$I_2 : \quad \vdash \square \quad (y_2 \equiv m_{1..3})$.

This property is symmetric to $I_1$.

$I_3 : \quad \vdash \square \quad \{[\ell_2 \wedge \sim m_2] \supset (t = 1)\}$.

Note that initially $\ell_2$ (i.e., $at\ell_2$) is false so that the implication is true. Potentially falsifying transitions are:

$\ell_1 \to \ell_2$: sets $t$ to 1.
$m_1 \to m_2$: makes $\sim m_2$ false.
$m_2 \to m_3$ while $\ell_2$: b y $I_1, y_1 = T$ so this transition is possible only when $t = 1$.

All other transitions trivially maintain the invariant.

$I_4 : \quad \vdash \square \quad \{[m_2 \wedge \sim \ell_2] \supset (t = 2)\}$.

Can be shown in a similar way.

We may now obtain the invariant ensuring mutual exclusion:

$I_5 : \quad \vdash \square(\sim \ell_3 \vee \sim m_3)$.

It is certainly true initially. The potentially falsifying transitions of this invariant are:

$\ell_2 \to \ell_3$ while $m_3$: but then $y_2 = T$ (by $I_2$) and $t = 1$ (by $I_3$), so that this transition is impossible.

$m_2 \to m_3$ while $\ell_3$: impossible, because $y_1 = T$ (by $I_1$) and $t = 2$ (by $I_4$).

Thus mutual exclusion has been formally proved.

## 5. LIVENESS PROPERTIES

We start by developing a proof rule which is more convenient to apply than the JUST rule.

<div style="border:1px solid">

(J-EVNT) - - The Just Eventuality Rule

Let $\varphi$ and $\psi$ be two state formulas and $T^J$ a justice set

    A.   $\vdash$  Every $\tau \in T$ leads from $\varphi$ to $\varphi \vee \psi$

    B.   $\vdash$  Every $\tau \in T^J$ leads from $\varphi$ to $\psi$

    C. $\vdash \varphi \supset \left(\psi \quad \vee \quad Enabled(T^J)\right)$

$$\overline{\qquad \vdash \varphi \supset \varphi \, \mathcal{U} \, \psi \qquad}$$

</div>

A similar rule exists for fairness. The rule can easily be derived from the JUST rule since by premise C every computation having in it a $\varphi$ which is not followed by a $\psi$, will have $T^J$ continuously enabled. This by the JUST rule implies $\varphi \mathcal{U} \psi$.

Let us apply the EVNT rule to our sample mutual exclusion program (Fig. 0). Take for example,

$$\varphi = \varphi_1 : \quad at\,\ell_2 \wedge at\,m_2 \wedge (t = 2) \wedge (y_1 = T) \wedge (y_2 = T)$$

$$\psi = \varphi_0 : \quad at\,\ell_3$$

Clearly the only transitions enabled on a state satisfying $\varphi_1$ are $\ell_2 \to \ell_3$ and $m_2 \to m_2$. Consequently every transition leads from $\varphi_1$ to $\varphi_1 \vee \psi$. Taking $T^J$ to be $P_1$, i.e., all transitions within $P_1$, we have premises A and B obviously satisfied. Also $\varphi_1$ implies that $\ell_2 \to \ell_3$ and hence $P_1$ is enabled. Thus we obtain $\vdash \varphi_1 \supset (\varphi_1 \mathcal{U} \varphi_0)$. From this we can certainly obtain

$$\vdash \varphi_1 \supset \Diamond \varphi_0$$

since $p \mathcal{U} q$ implies $\Diamond q$.

Next, let us take

$$\varphi = \varphi_2 : \quad at\,\ell_2 \wedge at\,m_1 \wedge (y_1 = T) \wedge (y_2 = T)$$

$$\psi = \varphi_1 \vee \varphi_0.$$

We now take $T^J$ to be $P_2$. Certainly, the only transitions possibly enabled under $\varphi_2$ are $\ell_2 \to \ell_2$, $\ell_2 \to \ell_3$ and $m_1 \to m_2$. The first transition preserves $\varphi_2$. The second transition leads from $\varphi_2$ to $\varphi_0$. The third transition which is guaranteed to be enabled under $\varphi_2$, leads from $\varphi_2$ to $\varphi_1$. Thus every transition leads from $\varphi_2$ to $\varphi_1 \vee \varphi_0$. We conclude $\vdash \varphi_2 \supset \Diamond(\varphi_1 \vee \varphi_0)$. From this we may conclude by temporal reasoning and the previously established $\vdash \varphi_1 \supset \Diamond \varphi_0$ that

$$\vdash \varphi_2 \supset \Diamond \varphi_0.$$

We may proceed and define additional $\varphi_j$, $j = 3, \ldots, 6$, such that for each $j$, $\vdash \varphi_j \supset \Diamond(\bigvee_{k<j} \varphi_k)$ which eventually leads to $\vdash \varphi_j \supset \Diamond \varphi_0$. This proof strategy of constructing a finite chain of assertions, each eventually leading to an assertion of lower index can be summarized by:

(CJIAIN) — The Chain Reasoni ng Proof Principle

Let $\varphi_0, \varphi_1, \ldots, \varphi_r$ be a sequence of state formulas.

A.  $\vdash$ Every $\tau \in T$ leads from $\varphi_i$ to $\bigvee\limits_{j \leq i} \varphi_j$,

B.  For every $i > 0$ there exists a justice set $T^J = T_i^J$ such that

  $\vdash$ Every $\tau \in T_i^J$ leads from $\varphi_i$ to $\bigvee\limits_{j < i} \varphi_j$

C.  For every $i > 0$ and $T_i^J$ as above:

  $\vdash \varphi_i \supset [ (\bigvee\limits_{j < i} \varphi_j) \vee Enabled(T_i^J)]$

---

$\vdash (\bigvee\limits_{i = 0}^{r} \varphi_i) \supset \Diamond \varphi_0$

The scheme of a proof according to the CHAIN principle is best presented in a form of a diagram. In this diagram we have a node for each $\varphi_i$. For each transition $\tau$ leading from a state satisfying $\varphi_i$ to a state satisfying $\varphi_j$ with $j \neq i$ (and hence by A, $j < i$) we draw an edge from $\varphi_i$ to $\varphi_j$. This edge is labelled by the appropriate justice set to which the transition belongs. Edges belonging to the justice set which is known by premise C to be enabled in $\varphi_i$ are drawn as double edges. For example, Fig. 1 contains a proof diagram for proving t- $at\,\ell_1 \supset 0$ $at\,\ell_3$ for the mutual exclusion program. By the CHAIN rule we actually proved $\vdash (\bigvee_{i=0}^{6}\varphi_i) \supset 0$ $at\,\ell_3$, but since $\varphi_6$ is $at\,\ell_1$ this establishes the desired result. The diagram representation of the CHAIN rule resembles closely the proof lattice advocated in [OL] for proving liveness properties.

In the application of the CHAIN rule we may freely usc any previously derived invariances of the program. Thus, if $\vdash \square I$ is any previously derived invariance, we may use $\varphi_i \wedge I$ instead of $\varphi_i$ to establish any of Lhe premises. This amounts to considering the sequence $\varphi_0 \wedge I, \ldots, \varphi_r \wedge I$ instead of the original sequence of assertions. Thus in the diagram (Fig. 1) we did not have an assertion corresponding to $(\ell_3, m_3)$ since by the previously established invariances such a situation is impossible, in particular no transition could lead from $I \wedge \varphi_4$ to $(\ell_3, m_3)$. Similarly no transition from $(\ell_2, ml)$ to $\ell_3$ has been drawn in view of $I_3$.

The chain reasoning principle assumed a finite number of links in the chain. It is quite adequate for finite state programs, i.e., programs where the variables range over finite domains. However, once we consider programs over infinite domains, such as Lhe integers, it is no longer sufficient to consider only finitely many assertions. In fact, sets of assertions of quite high cardinality arc needed. The obvious generalization to infinite sets of assertions is to consider a single state assertion $\varphi(\alpha, s)$, parametrized by a parameter $\alpha$ taken from a well-founded ordered set $(A, \prec)$. Obviously, an important feature of our chain of assertions is that program transitions led from $\varphi_i$ to $\varphi_j$ with $j < i$. This property can also be stated for an arbitrary well-founded ordering. Thus a natural generalization of the chain reasoning rule is the following:

In order to obtain a complete rule for liveness properties we have to treat the parametrized assertion $\varphi(\alpha, s)$ as an auxiliary assertion:

We refer the reader to [LPS] for a completeness proof of the LIVE principle. Completeness here means that given two state properties $q$ and $\psi$ such that $q \supset \Diamond \psi$ is a valid statement over all the computations of the program $P$, it is always possible to find state predicates p, $\varphi(\alpha, s)$ with $\alpha \in A$ and $(A, \prec)$, $h$ as in WELL that satisfy premises A to E. Note that premise D requires preliminary derivation of the invariance of $p$ which can be done using the INV rule.

## 6. PRECEDENCE PROPERTIES

As a key operator in expressing and establishing precedence properties we take the weak until operator, $\mathcal{U}$, to which we will refer here as the unless operator.

The *unless* operator may be defined in terms of the standard *until* operator as:

$$p \, \mathcal{U} \, q \equiv \Box p \vee (p \, \mathcal{U} \, q).$$

Thus, in contrast to $p \mathcal{U} q$ it does not require that $q$ eventually happen. But in the case that $q$ never happens $p$ is required to hold forever.

Even though it is introduced here as a derived operator, it can be adopted as the basic operator for establishing precedence properties. This is because both the *until* and precede operators can be expressed in terms of the unless operator:

$$p \, \mathcal{U} \, q \; \equiv \; (p \, \mathcal{U} \, q) \; \wedge \; \Diamond q$$

$$p \, \mathcal{P} \, q \; \equiv \; (\sim q) \, \mathcal{U} (p \, \wedge \, \sim q).$$

We can also express the nested *until* operator by considering the nested unless operator. Let $\psi_r, \psi_{r-1}, \ldots, \psi_1, \psi_0$ be a sequence of formulas then

$$\psi_r \, \mathcal{U} \, \psi_{r-1} \, \mathcal{U} \ldots \psi_1 \, \mathcal{U} \, \psi_0 \; \equiv \; \psi_r \, \mathcal{U} \left( \psi_{r-1} \, \mathcal{U} \left( \ldots (\psi_1 \, \mathcal{U} \, \psi_0) \right) \ldots \right)$$

holds on a sequence $\sigma = s_0, s_1, \ldots$ if there exists a sequence of indices $0 = i_r \leq i_{r-1} \leq \ldots \leq i_1 \leq i_0 \leq \omega$ such that for every $\ell > 0$ and j, $i_\ell \leq j < i_{\ell-1}, \psi_\ell$ holds on

$$\sigma^{(j)} \; = \; s_j, s_{j+1}, \ldots$$

and if $i_0 < \omega$ then $\psi_0$ holds on $\sigma^{(i_0)}$. Note that some of the $i_\ell$ may be equal to one another, and also to $\omega$ in which case some of the $\psi_\ell$ hold in empty periods.

An alternative description is that $\psi_r \, \mathcal{U} \ldots \psi_1 \, \mathcal{U} \, \psi_0$ holds on $\sigma$ iff either $\sigma$ satisfies $\psi_r \, \mathcal{U} \ldots \psi_1 \, \mathcal{U} \, \psi_0$ or for some j, $0 < j \leq r, \sigma$ satisfies $\psi_r \, \mathcal{U} \ldots \psi_{j+1} \, \mathcal{U} \, \Box \; \psi_j$. In the case j = r, $\sigma$ satisfies $\Box \, \psi_r$.

Then we can express the nested *until* by an extension of the previous formula for a simple *until:*

$$\psi_r \, \mathcal{U} \, \psi_{r-1} \, \mathcal{U} \ldots \psi_1 \, \mathcal{U} \, \psi_0 \equiv \left( \psi_r \, \mathcal{U} \, \psi_{r-1} \, \mathcal{U} \ldots \psi_1 \, \mathcal{U} \, \psi_0 \right) \wedge \Diamond \, \psi_0.$$

Let us justify this equivalence. The, direction in which the nested *until* implies the nested unless and the eventual occurrence of $\psi_0$ is obvious. Let us therefore consider the other direction.

Assume that $\psi_r \, \mathcal{U} \ldots \psi_1 \, \mathcal{U} \, \psi_0$ and $\Diamond \, \psi_0$ both hold on a sequence $\sigma$. By the interpretation of nested unless there exists a partition:

$$0 \; = \; i_r \leq i_{r-1} \leq \ldots \leq i_1 \leq i_0 \leq \omega$$

such. that $\psi_\ell$ holds between $i_\ell$ and $i_{\ell-1}$ for $\ell > 0$ and $\psi_0$ holds at $i_0$ if it is finite. Since $\psi_0$ must occur somewhere in $\sigma$ let j be the minimal index such that $\psi_0$ holds on $\sigma^{(j)}$. If $j = i_0 < \omega$, then the same partition justifies $\psi_r \, \mathcal{U} \ldots \psi_1 \, \mathcal{U} \, \psi_0$ on $\sigma$. Otherwise there exists some $\ell$ such that $i_\ell \leq j < i_{\ell-1}$. In this case the partition up to $i_\ell$ and then j justifies $\psi_r \, \mathcal{U} \ldots \psi_\ell \, \mathcal{U} \, \psi_0$ from which

$$\psi_r \, \mathcal{U} \ldots \psi_\ell \, \mathcal{U} \, \psi_{\ell-1} \ldots \psi_1 \, \mathcal{U} \, \psi_0$$

follows by letting $\psi_{\ell-1}, \ldots, \psi_1$ hold over empty periods.

Thus, expressively at least, the unless operator seems to be an appropriately basic operator. But we claim that the choice of the unless operator is appropriate on proof theoretic grounds as well. By inspecting the expression of *until* formulas in terms of unless formulas we find a resemblance

to the relation between the concepts of total and partial **correctness.** Total correctness, which is a liveness property, can be expressed as the conjunction of partial correctness, which is an invariance properly, and termination, which is another liveness property but simpler than the original. In quite the same way we can express the *until* properly as a conjunction of an unless property, which we regard as extended invariance property and the simpler liveness property $0 \ \psi_0$.

In practice, if we want a single proof principle that will cover properties of the following three su bclasses

$$(a) \quad \varphi \ \supset \ (p \, \mathcal{U} \, q)$$

$$(b) \quad \varphi \ \supset \ (p \, \mathcal{P} \, q)$$

$$(c) \quad \varphi \ \supset \ (p \, \mathcal{U} \, q)$$

then the unless operator is a good choice.

In order to establish (a) we establish separately

$$\vdash \ (\varphi \ \supset \ p \, \mathcal{U} \, q) \qquad \text{and} \qquad \vdash \ \varphi \ \supset \ \Diamond q,$$

which are implied by (a). The first will be established by using the unless proof principle. The second is a liveness property and can be established by the **WELL** rule or its extensions.

Similarly in order to establish (b) it is sufficient to establish $\varphi \supset (\tilde{p} \, \mathcal{U} \, \tilde{q})$ where $\tilde{p}$ is $\sim q$ and $\tilde{q}$ is $p \wedge \sim q$.

We could not have used the *until* operator in a similar role, i.e., reducing proofs of proper ties of the subclasses (b) and (c) to those of (a). This is for example because if $\varphi \supset (p \, \mathcal{U} \, q)$ is a valid statement, then certainly so is $\varphi \supset (\text{Cl } p \vee (p \mathcal{U} q))$, but it does not imply that either $\varphi \supset \text{Cl } p$ or $\varphi \supset (p \mathcal{U} q)$ are valid statements. Proving precede statements would cause similar problems.

The fact that the weak form of the *until* operator is more basic than its strong form seems to have been intuitively sensed in [L2] where a *while* operator is introduced which is equivalent to $p \, \mathcal{U} \sim q$.

Consequently, we will proceed by developing proof principles for the unless operator $\mathcal{U}$. We begin by formulating a core rule:

---

(CORE-u) — core Rule for Unless Properties

Let $\varphi_r, \varphi_{r-1}, \ldots, \varphi_0$ be state formulas

A. For every $i > 0$,

$\vdash$ Every $\tau \in T$ leads from $\varphi_i$ to $\displaystyle\bigvee_{j \leq i} \varphi_j$

$$\rule{8cm}{0.4pt}$$

$$\vdash \ \left(\bigvee_{i=0}^{r} \varphi_i\right) \ \supset \ (\varphi_r \, \mathcal{U} \, \varphi_{r-1} \, \mathcal{U} \ldots \varphi_1 \, \mathcal{U} \, \varphi_0)$$

---

Let $\sigma$ be a computation whose first state $s_0$ satisfies $\varphi_j$ for some $0 \leq j \leq r$. Assume first that $j > 0$. Define $i_r = i_{r-1} = \ldots = i_j = 0$. By premise A, $s_1$ must satisfy some $\varphi_\ell$ for $\ell \leq j$. If

$\ell = j$ we proceed until we find an $s_k$ that satisfies $\varphi_\ell$ for $\ell < j$. If we never find such a state we may take $i_{j-1} = \ldots = i_0 = \omega$. Otherwise we take $i_{j-1} = . , . = i_\ell = k$ and proceed similarly beyond $s_k$ unless $\ell = 0$. This construction shows that if $s_0$ satisfies $\varphi_j$ for some j then $\sigma$ satisfies $\varphi_r \, \mathfrak{U} \ldots \, \mathfrak{U} \varphi_0$. The case j = $\mathbf{0}$ is even simpler.

We can make a complete rule out of the CORE-U rule by strengthening the preconditions and weakening the post conditions.

---

(UNLS) -- Complete Rule for Unless Properties

Let $\varphi_r, \ldots,$ P O , $\psi_r, \ldots,$ $\psi_0, p, q$ be state formulas such that:

    A. For every i > 0,

        I- Every $\tau \in T$ leads from $\varphi_i \wedge p$ to $\displaystyle\bigvee_{j \leq i} \varphi_j$

    B. $\vdash \; \Box p$

    C. $\vdash \; (q \wedge p) \supset \left( \displaystyle\bigvee_{i=0}^{r} \varphi_i \right)$

    D. For every i, $0 \leq i \leq r$

        $\vdash \; (\varphi_i \wedge p) \supset \psi_i$

---

    $\vdash \; q \supset (\psi_r \, \mathfrak{U} \psi_{r-1} \, \mathfrak{U} \ldots \psi_1 \, \mathfrak{U} \psi_0)$

---

Let us consider the application of this rule to the analysis of the mutual exclusion algorithm. We take (the $\varphi_i$'s refer to the assertions in **Fig.** 1):

$q :$    $at\,\ell_2$

$\tilde{\varphi}_0 = \; \psi_0 \; : \; at\,\ell_3$

$\tilde{\varphi}_1 = \varphi_{1..3} : \; \ell_2 \wedge [m_{0,1} \vee (m_2 \wedge (t = 2))]$

$\tilde{\varphi}_2 = \varphi_4 : \; \ell_2 \wedge m_3$

$\tilde{\varphi}_3 = \varphi_5 : \; \ell_2 \wedge m_2 \wedge (t = 1)$

$\psi_1 = \psi_3 = \sim m_3, \qquad \psi_2 = m_3$

$p$  —   the conjunction of all the invariants $I_0 \wedge \ldots \wedge I_5$

The diagram certainly establishes that, $\tilde{\varphi}_i$, i > 0, leads to $\displaystyle\bigvee_{j \leq i} \tilde{\varphi}_j$.

It is also easy to show that $(q \wedge p) \supset \left( \displaystyle\bigvee_{i=1}^{3} \tilde{\varphi}_i \right)$ and that $\tilde{\varphi}_i \supset \psi_i$ for i = 0, $\ldots$, 3. Thus we may conclude:

    $\vdash \; \ell_2 \supset (\sim m_3 \, \mathfrak{U} m_3 \, \mathfrak{U} \sim m_3 \, \mathfrak{U} \ell_3).$

18

This establishes the property of 1-bounded overtaking from $\ell_2$. This means that once $P_1$ is at $\ell_2, P_2$ may be at $m_3$ at most once before $P_1$ gets to his critical section at $\ell_3$.

An alternative derivation of the same result could have been achieved by taking the 'p's in the rule to be identical to the 'p's in the diagram. This leads to:

$$\vdash \quad \ell_2 \supset (\varphi_5 \, \mathcal{U} \, \varphi_4 \, \mathcal{U} \, \varphi_3 \, \mathcal{U} \, \varphi_2 \, \mathcal{U} \, \varphi_1 \, \mathcal{U} \, \varphi_0).$$

We may now use the collapsing theorem for the unless operator:

$$(p \, \mathcal{U} \, q \, \mathcal{U} \, r) \supset ((p \vee q) \, \mathcal{U} \, r)$$

to obtain:

$$\vdash \quad \ell_2 \supset (\varphi_5 \, \mathcal{U} \, \varphi_4 \, \mathcal{U} \, (\varphi_1 \vee \varphi_2 \vee \varphi_3) \, \mathcal{U} \, \varphi_0),$$

which is equivalent to the above after we replace each of the $\varphi_i$'s by the weaker $\psi_i$.

Having obtained 1-bounded overtaking from the point that $P_1$ is at $\ell_2$ we may inquire whether the same holds from the point that $P_1$ is at $\ell_1$. As the analysis shows in Fig. 2 the best we can hope for is 2-bounded overtaking. The diagram in Fig. 2 establishes

$$\vdash \quad \ell_1 \supset (\varphi_8 \, \mathcal{U} \, \varphi_{5..7} \, \mathcal{U} \, \varphi_4 \, \mathcal{U} \, \varphi_{1..3} \, \mathcal{U} \, \varphi_0)$$

from which 't-bounded overtaking is easily established.


## 7. COMPLETENESS OF THE UNLS RULE

Next we will show that the UNLS rule presented above is complete for establishing nested unless properties.


Proof:

Let $q, \psi_r, \ldots, \psi_0$ be state properties such that the statement $q \supset (\psi_r \, \mathcal{U} \, \psi_{r-1} \ldots \psi_1 \, \mathcal{U} \, \psi_0)$ is valid on all admissible computations. We will show that there exist state properties $p, \varphi_r, \ldots, \varphi_0$, which are first-order expressible over the integers, such that all the premises of the UNLS rule are satisfied.

As p we choose

$$p(s) \quad \equiv Acc(s) \equiv \quad \{\text{There exists an initialized computation containing } s\}.$$

Clearly p is an invariant of all admissible computations so that premise B is satisfied.

Let $\tilde{\sigma}$ be a finite segment of a computation, i.e., a finite sequence

$$\tilde{\sigma} \quad = \quad s_0 \xrightarrow{\tau_1} s_1 \xrightarrow{\tau_2} \ldots \xrightarrow{\tau_k} s_k$$

such that $s_{i+1} \in f_\tau(s_i)$ for each $i = 0, \ldots, k - 1$.

We say that $\tilde{\sigma}$ satisfies a temporal formula $w$ if $\tilde{\sigma}$'s infinite extension $s_0, s_1, \ldots, s_k, s_k, s_k, \ldots$ satisfies $w$.

Let $\sigma$ be a computation satisfying $\psi_r \, \mathfrak{U} \ldots \psi_1 \, \mathfrak{U} \psi_0$. It can be verified that any finite prefix of $\sigma$ is a computation segment that also satisfies $\psi_r \, \mathfrak{U} \ldots \psi_1 \, \mathfrak{U} \psi_0$.

Let us define now $\varphi_i$ for $i = 0, 1, \ldots, r$ by $\varphi_i(s) = true$ iff

      (a)   Every computation segment originating at s satisfies $\psi_i \, \mathfrak{U} \psi_{i-1} \ldots \psi_1 \, \mathfrak{U} \psi_0$

      (6)   The index i is the srnallest index for which (a) holds.

Let us show that the sequence of $\varphi_i$'s defined in this way satisfies premises A, C and D of the UNLS rule.

Consider first premise A. Let s be a state satisfying $\varphi_i$, for $i > 0$. Let $s'$ be a state such that s' $\in f_\tau(s)$. Consider any computation segment originating in s':

$$\tilde{\sigma}' : \quad s' \xrightarrow{\tau_1} s_1 \xrightarrow{\tau_2} \ldots \xrightarrow{\tau_k} s_k.$$

We can obtain from it a computation segment:

$$\tilde{\sigma} : \quad s \xrightarrow{\tau} s' \xrightarrow{\tau_1} s_1 \xrightarrow{\tau_2} \ldots \xrightarrow{\tau_k} s_k.$$

By our assumption about s, $\tilde{\sigma}$ must satisfy $\psi_i \, \mathfrak{U} \ldots \mathfrak{U} \psi_0$. It can be shown that due to $i > 0$, and the minimality of $i$ this implies that $\tilde{\sigma}'$ must also satisfy $\psi_i \, \mathfrak{U} \ldots \mathfrak{U} \psi_0$. Thus we have identified at least one index, $i$, such that clause (a) is satisfied for $i$ and s'. Let $j \geq 0$ now be the minirnal index satisfying (a) for s'. Then (b) is also satisfied and we have that s' satisfies $\varphi_j$ for $j \leq i$. This establishes premise A.

Next, consider premise C. Let s be a state satisfying $q$ and $p$. It is therefore an accessible state satisfying (I. By the assumption that $q \supset (\psi_r \, \mathfrak{U} \ldots \mathfrak{U} \psi_0)$ is a valid statement for all admissible computations, every computation originating in s saisfies $\psi_r \, \mathfrak{U} \ldots \mathfrak{U} \psi_0$. Consequently every computation segment originating in s satisfies $\psi_r \, \mathfrak{U} \ldots \mathfrak{U} \psi_0$. Thus, clause (a) of the defini tion of $\varphi_i$ is satisfied for $i = r$. Let $j$ be the minimal index satisfying clause (a). Then $\varphi_j(s)$ holds and $j \leq r$.

To show premise D, let s be a state saisfying $\varphi_i$. Consider first $i = 0$. The zero version of $\psi_i \, \mathfrak{U} \ldots \mathfrak{U} \psi_0$ is $\psi_0$ by itself. Since every finite computation segment originating in s must saisfy $\psi_0$ which is a state property, it follows that s satisfies $\psi_0$. Consider next, $i > 0$. Since $i$ was the minimal index satisfying clause (u), there must exist a computation segment $\sigma$ originating in s which satisfies $\psi_i \, \mathfrak{U} \ldots \mathfrak{U} \psi_0$ but not $\psi_{i-1} \, \mathfrak{U} \ldots \mathfrak{U} \psi_0$. Consequently the initial section of $\tilde{\sigma}$ satisfying $\psi_i$ must be non-empty and therefore s must satisfy $\psi_i$. Thus, we have $\varphi_i \supset \psi_i$.

We claimed that the $\varphi_i$'s defined above are first-order expressible over the integers. This is due to the fact that clause (u) refers only to *finite* computation segments. This is a direct consequence of the fact Lhat we deal with the *unless* operator. No similar first-order definition is possible for the *until* operator. ◢

## 8. DIRECT PROOFS OF UNTIL PROPERTIES

In spite of our recommendation of splitting a proof of *until* property into a proof of a similar unless property, followed by a liveness proof of $0\,\psi$, there are many cases in which an *until* property can be directly obtained by a small modification of the liveness proof. As we have seen both the CHAIN rule and the UNLS rule call for a sequence of assertions, such that the computation always lead from $\varphi_i$ to $\varphi_j$ with $j \leq i$. The CHAIN rule stipulates in addition a strict decrease under certain conditions. It is often the case that the same chain of assertions used in the CHAIN rule can be used to establish a nested *until*. In fact, in much the same way that we have justified the CHAIN rule we can with the same premises obtain a stronger result:

Taking $0 < p_1 < p_2 < \ldots < p_s = r$ be a partition of the index range $[0...r]$ into s contiguous segments, we may formulate the following chain principle for until properties:

---

(U-CHAIN) -- The Chain Rule for Until Properties

Let $\varphi_0, \varphi_1, \ldots, \varphi_r$ be a sequence of state formulas, and $0 < p_1 < p_2 < \ldots < p_s = r$ a partition of $[1...r]$.

A. $\vdash$ Every $\tau \in T$ leads from $\varphi_i$ to $(\bigvee_{j \leq i} \varphi_j)$ for $i = 1, \ldots, r$.

B. for every $i > 0$ there exists a justice set $T^J = T_i^J$ such that:

   $\vdash$ Every $\tau \in T_i^J$ leads from $\varphi_i$ to $(\bigvee_{j < i} \varphi_j)$

C. for $i > 0$ and $T_i^J$ as above:

   $\vdash \varphi_i \supset [\,(\bigvee_{j < i} \varphi_j) \vee Enabled(T_i^J)]$

---

$$\vdash (\bigvee_{i=0}^{r} \varphi_i) \supset \left[ (\bigvee_{j=p_s-1+1}^{p_s} \varphi_j)\, \mathcal{U}\, (\bigvee_{j=P_s-2+1}^{p_s-1} \varphi_j)\, \mathcal{U} \ldots (\bigvee_{j=1}^{p_1} \varphi_j)\, \mathcal{U}\, \varphi_0 \right]$$

The conclusion states that starting at a state that satisfies one of the $\varphi_i$'s, $i = 0, \ldots, r$, we are guaranteed to have a period in which $(\bigvee_{j=p_s-1+1}^{p_s} \varphi_j)$ continuously hold, followed by a period in which $(\bigvee_{j=p_s-2+1}^{p_s-1} \varphi_j)$ continously holds, etc., until finally $\varphi_0$ is realized. Any of these periods may be empty.

To justify the soundness of this conclusion we first prove it for the most refined partition possible, namely:

$$(*) \qquad (\bigvee^{r} \varphi_i) \supset (\varphi_r\,\mathcal{U}\,\varphi_{r-1}\,\mathcal{U}\,\varphi_{r-2}\,{}_{\mathcal{U}} \ldots \varphi_1\,\mathcal{U}\,\varphi_0).$$

This is proved in a way similar to the justification of the corresponding liveness principle. We show

by induction on n, $n = 0, 1, \ldots, r$, that

$$\vdash \left( \bigvee_{i=0}^{n} \varphi_i \right) \supset (\varphi_n \mathcal{U} \varphi_{n-1} \mathcal{U} \ldots \varphi_1 \mathcal{U} \varphi_0).$$

For $n = 0$ we have $\vdash \varphi_0 \supset \varphi_0$ from which follows trivially

$$\vdash \varphi_0 \supset \varphi_0 \mathcal{U} \varphi_0.$$

Assume that the statement $(*)$ above has been proved for a certain $n$ and consider its proof for $n + 1$.

Consider the EVNT rule with $\varphi = \varphi_{n+1}, \psi = \left( \bigvee_{i=1}^{n} \varphi_i \right)$. As shown in the proof of the liveness case all 'the premises of the EVNT rule are satisfied. Consequently we may conclude:

$$\vdash \varphi_{n+1} \supset \varphi_{n+1} \mathcal{U} \left( \bigvee_{i=1}^{n} \varphi_i \right).$$

By the induction hypothesis and the monotonicity of the $\mathcal{U}$ operator this yields

$$\vdash \varphi_{n+1} \supset (\varphi_{n+1} \mathcal{U} \varphi_n \mathcal{U} \ldots \varphi_1 \mathcal{U} \varphi_0).$$

Due to $\vdash v \supset (u \mathcal{U} v)$, the induction hypothesis can also be written as

$$\vdash \left( \bigvee_{i=0}^{n} \ldots \right) \supset (\varphi_{n+1} \mathcal{U} \varphi_n \mathcal{U} \ldots \varphi_1 \mathcal{U} \varphi_0).$$

Taking the disjunction of the last two statements gives

$$\vdash \left( \bigvee_{i=0}^{n+1} \varphi_i \right) \supset (\varphi_{n+1} \mathcal{U} \varphi_n \mathcal{U} \ldots \varphi_1 \mathcal{U} \varphi_0),$$

which is the required statement $(*)$ for n + 1.

Consider now a coarser partition:

$$0 < p_1 < p_2 < \ldots < p_s = r.$$

By consecutively merging any. two contiguous assertions that fall into the same cell, using the collapsing rule:

$$\vdash \left( \varphi_{i+1} \mathcal{U} (\varphi_i \mathcal{U} \varphi) \right) \supset \left( (\varphi_{i+1} \vee \varphi_i) \mathcal{U} \varphi \right),$$

we obtain the coarser conclusion:

$$\vdash \left( \bigvee_{i=0}^{r} \varphi_i \right) \supset \left( \left( \bigvee_{j=p_{s-1}+1}^{p_s} \varphi_j \right) \mathcal{U} \left( \bigvee_{j=p_{s-2}+1}^{p_s-1} \varphi_j \right) \mathcal{U} \ldots \left( \bigvee_{j=1}^{p_1} \varphi_j \right) \mathcal{U} \varphi_0 \right). \quad \blacksquare$$

22

In our mutual exclusion program, by reference to Fig. 1 it is easy to use the U-CHAIN rule and obtain:

$$\ell_2 \supset (\varphi_5 \, \mathcal{U} \, \varphi_4 \, \mathcal{U} \, \varphi_{1..3} \, \mathcal{U} \, \varphi_0),$$

from which the 1-bounded overtaking from $\ell_2$ is obtained by the monotonicity of the *until* operator (i.e., replacing formulas by weaker formulas).

A natural extension of the U-CHAIN rule to programs that require infinite chains of assertions uses again well-founded ordered sets.

Let $(A, \prec)$ be a well-founded ordered set. We require however that the ordering is total (or linear). That is, for every two distinct elements, $\alpha_1, \alpha_2 \in A$ either $\alpha_1 \prec \alpha_2$ or $\alpha_2 \prec \alpha_1$.

---

(U-WELL) — Well-Founded Until Rule

Let $(A, \prec)$ be a well-founded totally ordered set.

Let $\varphi(\alpha) = \varphi(\alpha, s)$ be a parametrized state formula.

Let $h : A \to J$ be a helpfulness function identifying for each $\alpha \in A$ the helpful justice set $h(a) \in J$.

Let $\alpha_1 \prec \alpha_2 \prec \ldots \prec \alpha_s$ be a *finite* sequence of elements of A.

    A. $\vdash$ Every transition $\tau \in T$ leads from

$$\varphi(\alpha) \quad \text{to} \quad \psi \vee \exists \beta \big( (\beta \preceq \alpha) \wedge \varphi(\beta) \big)$$

    B. $\vdash$ Every transition $\tau \in h(a)$ leads from

$$\varphi(\alpha) \quad \text{to} \quad \psi \vee \exists \beta \big( (\beta \prec \alpha) \wedge \varphi(\beta) \big)$$

    C. $\vdash \varphi(\alpha) \supset [\psi \vee \exists \beta \big( (\beta \prec \alpha) \wedge \varphi(\beta) \big) \vee Enabled(\, h(a)\,)]$

---

$\vdash \exists \alpha \big( (\alpha \preceq \alpha_s) \wedge \varphi(\alpha) \big) \supset$
$$\big[ \exists \beta \big( (\alpha_{s-1} \prec \beta \preceq \alpha_s) \wedge \varphi(\beta) \big) \, \mathcal{U}$$
$$\exists \beta \big( (\alpha_{s-2} \prec \beta \preceq \alpha_{s-1}) \wedge \varphi(\beta) \big) \, \mathcal{U} \, \ldots$$
$$\exists \beta \big( (\beta \preceq \alpha_1) \wedge \varphi(\beta) \big) \, \mathcal{U} \, \psi \big]$$

---

By a combination of the completeness of the WELL rule for liveness properties and the UNLS rule for *unless* properties we can extend the above rule to a complete rule for *until* properties.

## 9. DECISION PROCEDURES FOR FINITE STATE PROGRAMS

The question of whether a given program has a certain property expressed by a temporal formula, is in general highly undecidable. However, for a very important restricted class of programs, this question is decidable, namely for finite state programs. Finite state programs are programs whose variables range each over a finite domain. These programs generate only finitely many different states and a joint, finite transition diagram over these states can be constructed such that any computation is a maximal path in this finite directed graph. The literature abounds in. many special decision procedures for testing for deadlock situations, starvation, etc. on programs

represented by finite transition diagrams. All these are special cases of the general result which states that testing a temporal formula over a finite state program is decidable. The general decision procedure for testing a temporal formula $\varphi$ on a finite state program P consists in checking the implication $W_P \supset \varphi$ for general validity. In this implication $W_P$ is a formula characterizing all admissible computations of $P$. If $P$ is finite state then both $W_P$ and $\varphi$ may be represented *as propositional* temporal formulas. Consequently we test a propositional temporal formula for general validity. As shown in [PS], it can be done in time exponential in the size of $P$ and $\varphi$. This exponential time complexity has been a source of criticism of linear temporal logic in [CES].

In this section we show that when the temporal property $\varphi$ to be tested, falls into one of the property classes discussed here, then there exists an efficient decision procedure polynomial in the size of $P$ and $\varphi$ for testing $\varphi$ on P.

Let $P$ be a program consisting of m processes $P_1, \ldots, P_m$. Let each process $P_i$ be presented as transition diagram with set of nodes $L_i$. The program variables $y_1, \ldots, y_n$ assume values over finite domains $D_1, \ldots, D_n$ respectively. Then the state set $S$ of the program $P$ is the set of all possible tuples $\langle \ell_1, \ldots, \ell_m; \eta_1, \ldots, \eta_n \rangle$ with $\ell_i \in L_i, i = 1, \ldots, m$, and $\eta_j \in D_j$ for $j = 1, \ldots,$ n. Consequently

$$|S| \leq |L_1| \times \cdots \times |L_m| \times |D_1| \times \cdots \times |D_n|.$$

We construct for $P$ a joint transition diagram $T_P$ with $S$ as nodes, and an edge $s \xrightarrow{P_i} s'$ for every pair of states s, $s'$ and a transition $\tau$ in $P_i$ which leads from $s$ to $s'$.

In order to generate only accessible states we start from all states satisfying $\theta$ and include in $T_P$ only states which are derivable from states which are already included in $T_P$. Fig. 3 shows the diagram $T_P$ for the mutual exclusion algorithm. States in this diagram have the form $(\ell_i, m_j, t)$. We have not included the values of $y_1, y_2$ since in all accessible states they are uniquely determined by the location values $\ell_i$ and $m_j$. The initial state in this diagram is so.

We proceed to describe three algorithms which, for properties in each of the three classes, will determine whether a finite state program $P$ has this property. The algorithms will be linear in the size of $T_P$. Let us denote $N = |T_P|$.

## 10. TESTING INVARIANCES

Let the formula to be tested be of the form $q \supset \Box \, \varphi$. We can check whether all paths in $T_P$, and hence all admissible computations of $P$, satisfy $q \supset Cl \, \varphi$ by the following procedure:

PI:   Locate in $T_P$ all states which satisfy $q$. For each such state $s$ construct the transition diagram $T_P(s)$ which includes exactly all the states accessible from s. Check that each s' $\in T_P(s)$ satisfies $\varphi$.

If all these steps succeeded then $q \supset Cl \, \varphi$ is valid for $P$. We can organize the procedure so that it takes no more than $m \cdot N$ steps where $N = |T_P|$ and m is the number of processes and hence the maximal degree of $T_P$. This is because if $s_2 \in T_P(s_1)$ satisfies $q$ then $T_P(s_2) \subset T_P(s_1)$ and no separate check is needed for $s_2$ if we have already checked $T_P(s_1)$.

Consequently we have to access each state at most once, and then may have to explore each of its edges.

For checking invariances we may actually suggest a simpler procedure: mark in $T_P$ each state which is accessible from a q-state (a state saisfying 9). Then check that all the marked states satisfy $\varphi$. However the complexity of the two procedures is identical and the PI procedure above conforms better with the procedures presented below for the other classes.

We may for example apply PI to test for the invariance of $I_0$ to $I_5$ derived for the mutual exclusion. All these properties have the form $\Box \varphi$ so we may take $q = true$ and consider $T_P(s)$ for all accessible states. However since every accessible state $s \in T_P(s_0) = T_P$, it is sufficient to check that all states in $T_P$ satisfy $\varphi$.

Indeed we can easily check for example that there are no states in which $\ell_2, \sim m_2$ and $t \neq 1$ are all true. In other words every state in which both $\ell_2$ and $\sim m_2$ are true, i.e., $s_6, s_{19}$, also has $t = 1$ in it. This establishes $I_3$. Similarly, there is no accessible state in which both $\ell_3$ and $m_3$ hold, establishing $I_5$.

It is easy to prove:


Lemma:

A formula $q \supset \quad \Box \quad \varphi$ is valid for $P$ iff the procedure PI applied to $T_P$ succeeds.


## 11. TESTING LIVENESS

Let the formula to be tested be of the form $q \supset \Diamond \varphi$. Let $s \in T_P$ be an accessible state. Let $\pi = s_1, \ldots, s_k$ be a finite path in $T_P$. We say that $\pi$ is a non-$\varphi$ path if none of $s_1, \ldots, s_{k-1}$ satisfy $\varphi$. Note that $s_k$ is allowd to satisfy $\varphi$. We define $T_P(s, \varphi)$ to be the directed graph containing all states in $T_P$ which are accessible from $s$ by non-$\varphi$ paths. The graph $T_P(s, \varphi)$ can be efficiently constructed as follows:

(a)   Put   $s$ in $T_P(s, \varphi)$

(b)   For every s' $\in T_P(s, \varphi)$ which does not satisfy $\varphi$, add all the successors of $s'$ to $T_P(s, \varphi)$.

Let us decompose $T_P(s, \varphi)$ into maximal strongly connected components. It is known that when we consider edges between the components, it is always possible to order the components in a topological sorting order $K_1, \ldots, K_r$, such that if there is an edge from a node in $K_i$ to a node in $K_j$ then necessarily $i \leq j$. Components such that there are no edges leading out of them are called *terminal* components.

We suggest the following test for checking that all just computations in $T_P(s, \varphi)$ satisfy $\Diamond \varphi$:

$\varphi$-*Liveness* Test:

Decompose $T_P(s, \varphi)$ into a topologically sorted list of maximal strongly connected components: $K_1, \ldots, K_r$.

For each $i = 1, \ldots, r$ check:

(a)  If $K_i$ is terminal then it consists of a single node satisfying $\varphi$.

(b)  If $K_i$ is nonterminal, then there must exist a j, j = 1, . . . , $m$, such that every state $s \in K_i$ has a $P_j$ transition leading out of $K_i$.

Lemma:

All just computations in $T_P(s, \varphi)$ realize $0\ \varphi$ iff the $\varphi$-liveness test succeeds.

Proof:

Assume that the test succeeds. Let $\sigma$ be any maximal computation in $T_P(s, \varphi)$. By the ordering of the $K_1, \ldots K_r$, from a certain point on, the computation must be fully contained in a single component, $K_\ell$ say. If $K_\ell$ is terminal then the computation terminates once it has entered $K_\ell$, and the last state satisfies $\varphi$ by (a) above. If $K_\ell$ is not terminal then being contained in $K_\ell$ and by (b) it must be infinite, since no state in $K_\ell$ is terminal. Furthermore, no $P_j$ transition is ever taken once the computation has entered $K_\ell$, otherwise it would have left $K_\ell$. Consequently the computation is unjust, with respect to $P_j$. Thus all just computation must eventually realize $\varphi$.

Assume that the test fails. Then either there is a terminal component $K_i$ not satisfying $\varphi$, or there exists a nonterminal component $K_i$ not satisfying condition (6). In the first case we construct a computation $\sigma$ leading from $\sigma$ to $K_i$, and then either stopping if the state $s \in K_i$ is terminal or looping within $K_i$ in a loop that spans all of $K_i$. Since states within $K_i$ do not satisfy $\varphi$ (actually none of them does) this can be shown to be a just computation not realizing $\varphi$. In the second case, we construct again a computation $\sigma$ reaching $K_i$ and continuing in a loop spanning all the transitions within $K_i$. By violation of condition (b) every process $P_j$ that has not terminated yet has a $P_j$ transition internal to $K_i$. Thus by traversing all transitions in $K_i$, we generate a just computation which does not realize $\varphi$.

Note that the construction of $T_P$, its decomposition into strongly connected components and applying the liveness test are all linear in the size of $T_P$.

In order to check that $q \supset 0\ \varphi$ is valid for $P$ we could in principle take each $s \in T_P$ which satisfies $q$, construct $T_P(s, \varphi)$ and apply the $\varphi$-liveness test to it. But we can actually be more efficient as follows:

Let $s_1, \ldots, s_k$ be all the q-states in $T_P$. Construct $T_P(s_1, \varphi_1)$ and check it for $\varphi_1$-liveness, where

$$\varphi_1(s) = \varphi(s).$$

Next, construct $T_P(s_2, \varphi_2)$ and check it for $\varphi_2$-liveness, where

$$\varphi_2(s) = p(s) \ \lor \ s \in T_P(s_1, \varphi_1)$$

Thus in constructing $T_P(s_2, \varphi_2)$ we may stop the analysis once the computation enters $T_P(s_1, \varphi_1)$, since we already know that, all computations there realize $\varphi$.

In general we construct $T_P(s_i, \varphi_i)$ and check it for $\varphi_i$-liveness for $i = 1, \ldots, k$ where:

$$\varphi_i(s) = \varphi(s) \lor [s \in \bigcup_{j<i} T_P(s_j, \varphi_j)].$$

26

In this way we essentially consider each state at most once and the whole procedure becomes *linear* in $|T_P|$. $\blacksquare$

Let us apply this procedure for checking validity of $at\,\ell_1 \supset 0\;at\,\ell_3$ on the mutual exclusion program. We will check the following q-states:

$$s_{17} : (\ell_1, m_3, 2), \quad s_{12} : (\ell_1, m_0, 2), \quad s_{13} : (\ell_1, m_1, 2),$$

$$s_1 : (\ell_1, m_0, 1), \quad s_3 : (\ell_1, m_1, 1), \quad s_{16} : (\ell_1, m_2, 2).$$

In Fig. 4 we present $T_P(s_{17}, at\,\ell_3)$. In decomposing the graph we find that every component consists of exactly one node and a possible sorting order for them is:

$$s_{17}, \; s_{12}, \; s_{13}, \; s_{16}, \; s_{18}, \; s_{19}, \; s_4, \; s_5, \; s_6, \; s_8, \; s_9.$$

The terminal components are $s_5$ and $s_9$ and they both satisfy $at\,\ell_3$. For every other component we easily identify a helpful process leading out of the component. Thus $P_1$ is helpful for $\{s_{17}, \; s_{12}, \; s_{13}, \; s_{16}, s_4, s_8\}$ and $P_2$ is helpful for $\{s_{18}, s_{19}, s_6\}$.

Note that this diagram also took care of $s_{12}, s_{13}, s_{16}$. The next q-state not yet analyzed is $s_1$. We construct for it $T_P(s_1, \varphi_2)$ where $\varphi_2(s) = at\,\ell_3 \vee s \in T_P(s_{17}, \ell_3)$.

The corresponding diagram in Fig. 5 shows that all computations starting at $s_1$ or $s_3$ eventually must enter $T_P(s_{17}, at\,\ell_3)$. Consequently we conclude that $at\,\ell_1 \supset 0\;at\,\ell_3$ is valid for the program $P$.

## 12. TESTING UNLESS PROPERTIES

Let the formula to be tested be

$$q \supset (\varphi_r \,\mathfrak{U}\, \varphi_{r-1} \ldots \varphi_1 \,\mathfrak{U}\, \varphi_0).$$

Let $s \in T_P$ be an accessible q-state. Construct $T_P(s, \varphi_0)$ as before. We propose the following test for checking that all computations in $T_P(s, \varphi_0)$ satisfy w : $\varphi_r \,\mathfrak{U}\, \varphi_{r-1} \ldots \varphi_1 \,\mathfrak{U}\, \varphi_0$.

*w-Precedence Test:*

Decompose $T_P(s, \varphi_0)$ into a topologically sorted list of maximal strongly connected components: $K_1, \ldots, K_{\ldots}$ Proceeding from $K_r$ down to $K_1$, we try to assign each component $K_i$ a rank $\rho_i = \rho(K_i)$ as follows:

Let $\rho_i$ be the smallest $k \geq 0$ such that all states in $K_i$ satisfy $\varphi_k$ and that any component $K_j$, directly connected to $K_i, i \geq j$, has a lower or equal rank, i.e., $k \geq \rho_j$.

If we fail to rank some component $K_i$ then the test is said to fail, otherwise we say that it has succeeded.

## Lemma A:

If the w-precedence test succeeds, then all computations in $T_P(s, \varphi_0)$ satisfy w.

## Proof:

Assume that the test succeeded. Let $\varphi$ be any computation in $T_P(s, \varphi_0)$. Such a computation must progress through a finite chain of components $K_{i_1}, K_{i_2}, \ldots, K_{i_\ell}$, with $i_1 < i_2 < \ldots < i_\ell$. Thus it successively satisfies $\varphi_{\rho(K_{i_1})}, \varphi_{\rho(K_{i_2})}, \ldots \varphi_{\rho(K_{i_\ell})}$ with $\rho(K_{i_1}) \geq \rho(K_{i_2}) \geq \ldots \geq \rho(K_{i_\ell})$.

Obviously it satisfies w.

Let $K_i$ be any component. We say that we failed to assign $K_i$ the rank j if either $\rho_i > j$ or we failed to rank $K_i$ altogether.

## Lemma B:

If we failed to assign $K_i$ the rank j then for every $s \in K_i$ there exists a computation $\sigma = s \rightarrow \ldots$ (beginning in s) that does not satisfy

$$w_j = \varphi_j \, \mathfrak{U} \ldots \varphi_1 \, \mathfrak{U} \varphi_0.$$

## Proof:

We will prove the lemma by double induction, first on j = 0, 1, . . . and then for each j on $i = r, r-1, \ldots, 1$.

Consider first j = 0. Let $s \in K_i$ be any state in $K_i$. If s satisfies $\varphi_0$ then $K_i$ consists of s alone and has no successors. Correspondingly we could have defined $\rho(K_i) = 0$. Since we failed to assign 0 to $K_i$, s does not satisfy $\varphi_0$. Consequently any computation beginning in s falsifies $w_0 = \varphi_0$. This establishes the lemma for j = 0 and $K_1, \ldots, K_r$.

Consider now a j > 0 and assume by induction that the lemma has been proved for j − I and $K_i$ and also for j and each of $K_{i+1}, \ldots, K_r$. Let $s \in K_i$.

. There could be two distinct reasons why we failed to assign the rank j to $K_i$.

- There exists some state $s^1 \in K_i$ which does not satisfy $\varphi_j$. By the induction hypothesis there exists a computation $\sigma' = s^1, s^2, \ldots$ which does not satisfy $w_{j-1}$. We claim that $\sigma'$ also does not satisfy $w_j$. For $\sigma'$ to satisfy $w_j$ there must be a (possibly empty) prefix of $\sigma'$ continuously satisfying $\varphi_j$ followed by a suffix which satisfies $w_{j-1}$. Since $s^1$ falsifies $\varphi_j$, the prefix must be empty and the whole of $\sigma'$ must satisfy $w_{j-1}$ which contradicts the definition of $\sigma'$.

It only remains to obtain a similar computation starting from s, the arbitrarily specified state in $K_i$. If by chance $s = s^1$ then $\sigma'$ will do. Otherwise, since s and $s^1$ belong to the same strongly connected component there must exist a path $s = s_1, \ldots, s_m = s^1$ within $K_i$ connecting s to $s^1$. Consider the computation $\sigma = s, \ldots, s^1, s^2, \ldots$, i.e., the path

28

from $s$ to $s^1$ followed by $\sigma'$. Since no state in $K_i$ satisfies $\varphi_0, \sigma$ can satisfy $w_j$ only if $\sigma'$ does. Thus $\sigma$ falsifies $w_j$.

- The second case where we fail to assign j to $K_i$ is that there exists a $K_\ell$ directly connected to $K_i, i < \ell$, such that $\rho_\ell > j$ or more generally we failed to assign j to $K_\ell$. Thus there exists $s_i \in K_i$ and $s_\ell \in K_\ell$ such that

$$s_i \xrightarrow{P_k} s_\ell \quad \text{for some } P_k.$$

By strong connectedness there exists a (possibly empty) path connecting s to $s_i$ : s, $\ldots, s_i$. By the induction hypothesis since $\ell > i$ and we failed to assign j to $K_\ell$ there exists a computation $\sigma_\ell : s_\ell, s^2, \ldots$ which falsifies $w_j$. Consider now the computation

$$\sigma : s, \ldots, s_i, s_\ell, s^2, \ldots$$

The computation $\sigma$ consists first of the path from $s$ to $s_i$ within $K_i$, then the edge from $s_i$ to $s_\ell$ and then follows $\sigma_\ell$. Since the whole segment s, $\ldots, s_\ell$ does not contain a state satisfying $\varphi_0, \sigma$ can satisfy $w_j$ only if $\sigma_\ell$ does, which is impossible. Thus $\sigma$ falsifies $w_j$ as required. ∎

Let now $K_i$ be a component that was not ranked altogether. By the last lemma there exists a computation $\sigma = $ s, $s^2, s^3, \ldots$ with $s \in K_i$ such that $\sigma$ falsifies

$$w_r = \varphi_r \, \mathfrak{U} \ldots \varphi_1 \, \mathfrak{U} \varphi_0.$$

We can prefix $\sigma$ by a path leading from $s_0$ to $s$ and obtain a computation $\sigma_0 = s_0, \ldots, s, \ldots$ which fails to satisfy w,. We may combine Lemmas A and B to obtain:

## Corollary:

Given $T_P(s_0, \varphi_0)$, all so-initialized computations in $T_P(s_0, \varphi_0)$ satisfy

$$w = \varphi_r \, \mathfrak{U} \ldots \varphi_1 \, \mathfrak{U} \varphi_0$$

iff the $w$-precedence test succeeded.

## Proof:

In order to test the general implication $q \supset w$ on the entire $T_P$ diagram we proceed as follows:

Let $s_1, s_2, \ldots, s_k$ be all the q-states in $T_P$. Construct $T_P(s_1, \varphi_0)$ and test $\varphi_r \, \mathfrak{U} \ldots \varphi_1 \, \mathfrak{U} \varphi_0$ on it. Construct $T_P(s_2, \psi_2)$ where $\psi_2(s) = \varphi_0(s) \vee s \in T_P(s_1, \varphi_0)$.

Test $\varphi_r \, \mathfrak{U} \ldots \varphi_1 \, \mathfrak{U} \varphi_0$ on $T_P(s_2, \psi_2)$. In ranking the components we add the following rule:

If $K_i$ is a terminal component consisting of the single node $s \in T_P(s_1, \varphi_0)$, give $K_i$ the rank that $s$ (or the component containing $s$) has received in $T_P(s_1, \varphi_0)$.

In general we construct $T_P(s_i, \psi_i)$ where

$$\psi_i(s) = \varphi_0(s) \vee [s \in \bigcup_{j < i} T_P(s_j, \psi_j)] \qquad (\psi_1 = \varphi_0).$$

We then test $\varphi_r \, \mathfrak{U} \ldots \mathfrak{U} \, \varphi_0$ on $T_P(s_i, \psi_i)$ ranking any component consisting of $s \in T_P(s_j, \psi_j)$ for some $j < i$ according to the rank it received earlier.

Consequently the testing procedure is again linear in the size of $T_P$. To be precise, of complexity $r \cdot m \, \cdot \, |T_P|$.

To illustrate the procedure let us test the validity of the following unless property:

$$\ell_0 \supset (\ell_0 \, \mathfrak{U} \, m_3 \, \mathfrak{U} \sim m_3 \, \mathfrak{U} \, m_3 \, \mathfrak{U} \sim m_3 \, \mathfrak{U} \, \ell_3).$$

This property again expresses a certain kind of 2-bounded overtaking. However the reference point is when $P_1$ is at $\ell_0$. It states that from the time $P_1$ decides to leave $\ell_0$, $P_2$ may enter $m_3$ at most twice before $P_1$ enters $\ell_3$. Furthermore, actual 2-overtaking can take place only if $P_1$ on exiting $\ell_0$ finds $P_2$ in $m_3$ at precisely the same moment. If on exiting Co, $P_1$ find $P_2$ anywhere else then at most 1-overtaking can take place. In contrast with other unless properties considered before in this paper, this property is not an *until* property. The corresponding until property does not hold since when $P_1$ is at $\ell_0$ it is quite acceptable that it never gets out to achieve $\ell_3$.

We define

$$q \; = \; \varphi_5 \; : \; at \, \ell_0$$

$$\varphi_4 \; = \; \varphi_2 : u \, t \, m \, 3$$

$$\varphi_3 \; = \; \varphi_1 : \sim \; at \, m_3$$

$$\varphi_0 \; = \; at \, \ell_3$$

Accessible q-states in $T_P$ are:

$$s_{15} : \; (\ell_0, m_3, 2), \quad s_{10} : (\ell_0, m_0, 2), \quad s_{11} : \; (\ell_0, m_1, 2),$$

$$s_{14} : \; (\ell_0, m_2, 2), \quad s_0 \; : \; (\ell_0, m_0, \; 1), \quad s_2 \; : \; (\ell_0, m_1, 1).$$

In Fig. 6 we have $T_P(s_{15}, \varphi_0)$. Its component decomposition gives the following topologically sorted list of components:

$$K_1 \; = \; \{s_{15}, s_{10}, s_{11}, s_{14}\}, \{s_{17}\}, \{s_{12}\}, \{s_{13}\}, \{s_{16}\}, \{s_{18}\}, \{s_{19}\}, \{s_4\}, \{s_5\}, \{s_6\}, \{s_8\}, \{s_9\}.$$

Going backwards we assign the following ranks:

$$\rho_i = 0 \; \text{for} \; i \in \{5, 9\}$$

$$\rho_i = 1 \; \text{for} \; i \in \{8, 6, 4\}$$

$$\rho_i = 2 \; \text{for} \; i \; = \; 19$$

$$\rho_i = 3 \; \text{for} \; i \in \{18, 16, 13, 12\}$$

$$\rho_i = 4 \; \text{for} \; i = 17$$

$$\rho(K_1) = 5$$

This shows that the desired unless property actually holds for the q-states $s_{15}, s_{10}, s_{11}, s_{14}$.

Next let us consider $T_P\big(s_0, [\varphi_0(s) \vee s \in T_P(s_{15}, \varphi_0)]\big)$. It is given in Fig. 7. All the terminal nodes belong to the previous diagram and their ranks have been listed. We may proceed to rank the unranked states in $T_P(s_0, \psi_2)$.

We define

$$\rho_i = 3 \ \text{f o r} \ i \in \{1, 3\},$$

and

$$\text{P}_i = 5 \ \text{f o r} \ i \in \{0, 2\}.$$

Thus, all q-states have been successfully ranked, and the unless property:

$$\ell_0 \supset (\ell_0 \ \mathfrak{U} m_3 \ \mathfrak{U} {\sim} m_3 \ \mathfrak{U} m_3 \ \mathfrak{U} {\sim} m_3 \ \mathfrak{U} \ell_3).$$

has been established. We obviously cannot do better since the computation:

$$s_{15} \rightarrow s_{17} \rightarrow s_{12} \rightarrow s_{13} \rightarrow s_{16} \rightarrow s_{18} \rightarrow s_{19} \rightarrow s_4 \rightarrow s_5$$

demonstrates 2-overtaking.


Acknowledgement:

## 13.   REFERENCES

[CES]   Clarke, E.M., E.A. Emerson, and A.P. Sistla, "Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specifications: A Practical Approach," Proc. of the IEEE Conf. on Foundations of Computer Science, Chicago (1982).

[K] Keller, R.M., "Formal verification of parallel programs," CACM, Vol. 19, No. 7 (July 1976), pp. 371–384.

[L1] Lamport, I,., "Proving the Correctness of Multiprocess Programs," IEEE Trans. Soft. Eng. SE-3, 2 (Mar. 1977), pp. 125-143.

[L2] Lamport, L., " 'Sometime' is Sometimes "Not Never': On the Temporal Logic of Programs," 7th Annual ACM Symposium on Principles of Programming Languages (1980), pp. 174-185.

[LPS] Lehmann, D., A. Pnueli, and J. Stavi, "Impartiality, justice and fairness: the ethics of concurrent termination," in Automata Languages and Programming, Lecture Notes in Computer Science 115, Springer Verlag (1981), pp. 264–277.

[MP1]   Manna, Z. and A. Pnueli, "Verification of Concurrent Prorams: The Temporal Frame-work," in *The Correctness Problem in Computer* Science (R.S. Boyer and J S. Moore, eds.), International Lecture Series in Computer Science, Academic Press, London (1982), pp. 215-273.

[MP2]   Manna, Z. and A. Pnueli, "Verification of Concurrent Programs: Temporal Proof Principles," Proc. of the Workshop on Logic of Programs (D. Kozen, cd.), Yorktown-Heights, N.Y. (1981). Springer-Verlag Lecture Notes in Computer Science 131, pp. 200-252.

[MP3]   Manna, Z. and A. Pnucli, "Verification of Concurrent Programs: Proving Eventualities by Well-Founded Ranking," TOPLAS (1983, to appear).

[MP4]   Manna, Z. and A. Pnueli, "Verification of Concurrent Programs: a Temporal Proof System," Proc. 4th School on Advanced Programming, Amsterdam, Holland (June 1982).

[MP5]   Manna, Z. and A. Pnueli, "How to Cook a Temporal Proof System for Your Pet Language," in the Proc. of the Symposium on Principles of Programming Languages, Austin, Texas (Jan. 1983).

[OL]   Owicki, S. and L. Lamport, "Proving Liveness Properties of Concurrent Programs," ACM Transactions on Programming Languages and Systems, Vol. 4, No. 3 (July 1982), pp. 455-495.

[OG]   Owicki, S. and D. Gries, "An Axiomatic Proof technique for Parallel Programs," Acta Informatica, Vol. 6, No. 4 (1976), pp. 319-340.

[Pe]   Peterson, G.L., "Myths about the Mutual Exclusion Problem," Information Processing Letters, Vol. 12, No. 3 (June 1981), pp. 115-116.

[PS]   Pnucli, and A., Sherman R., "Semantic Tableau for Temporal Logic," Technical Report, CS81-21, The Weizmann Institute (Sept. 81).
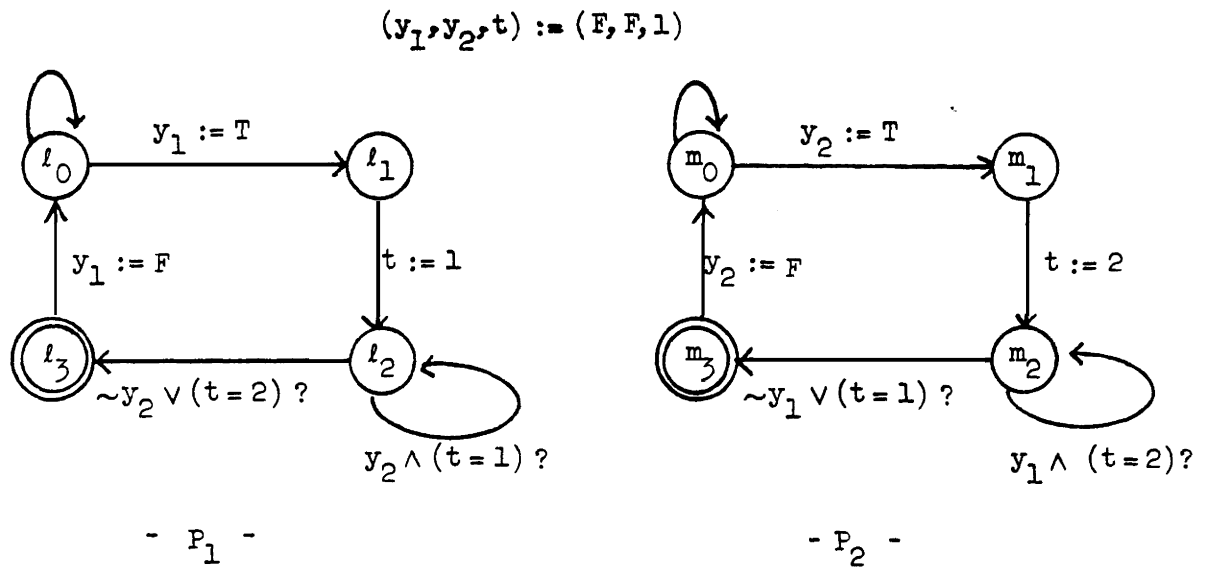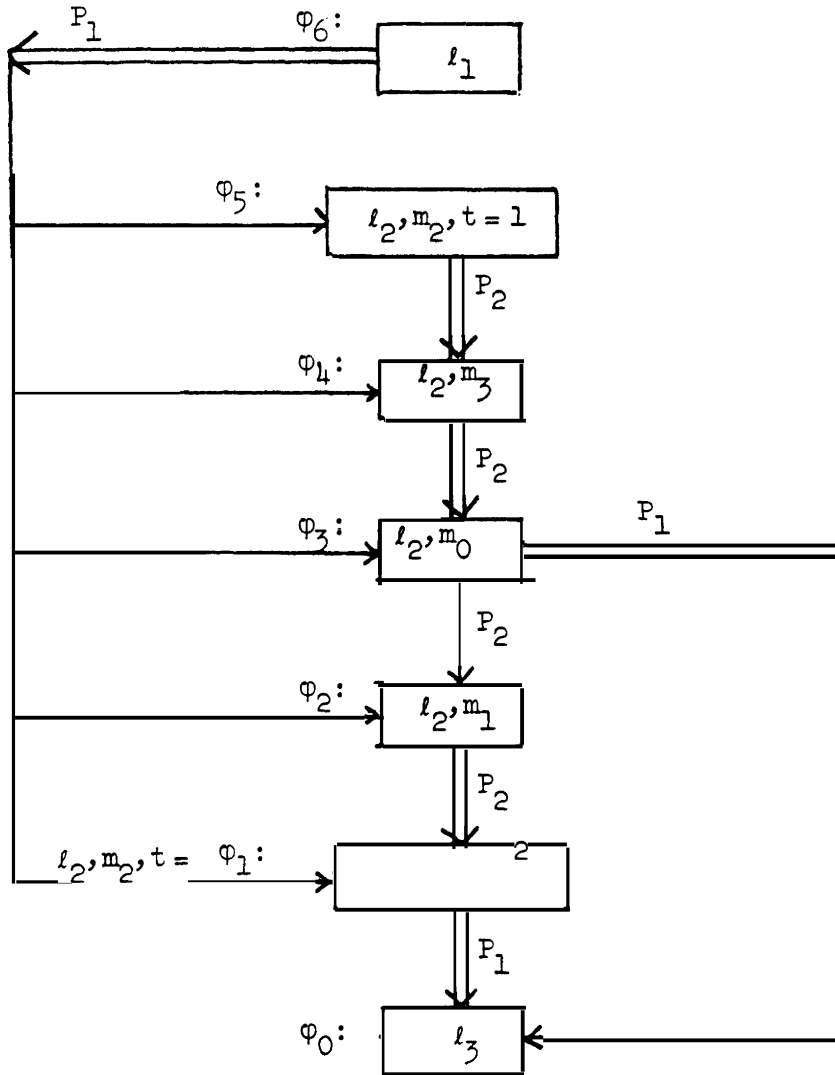
$$(y_1, y_2, t) := (F, F, 1)$$



- P_1 -

- P_2 -

Figure 0



Fig. 1.   Proof Diagram for $\vdash \ell_1 \supset \diamondsuit \ell_3$
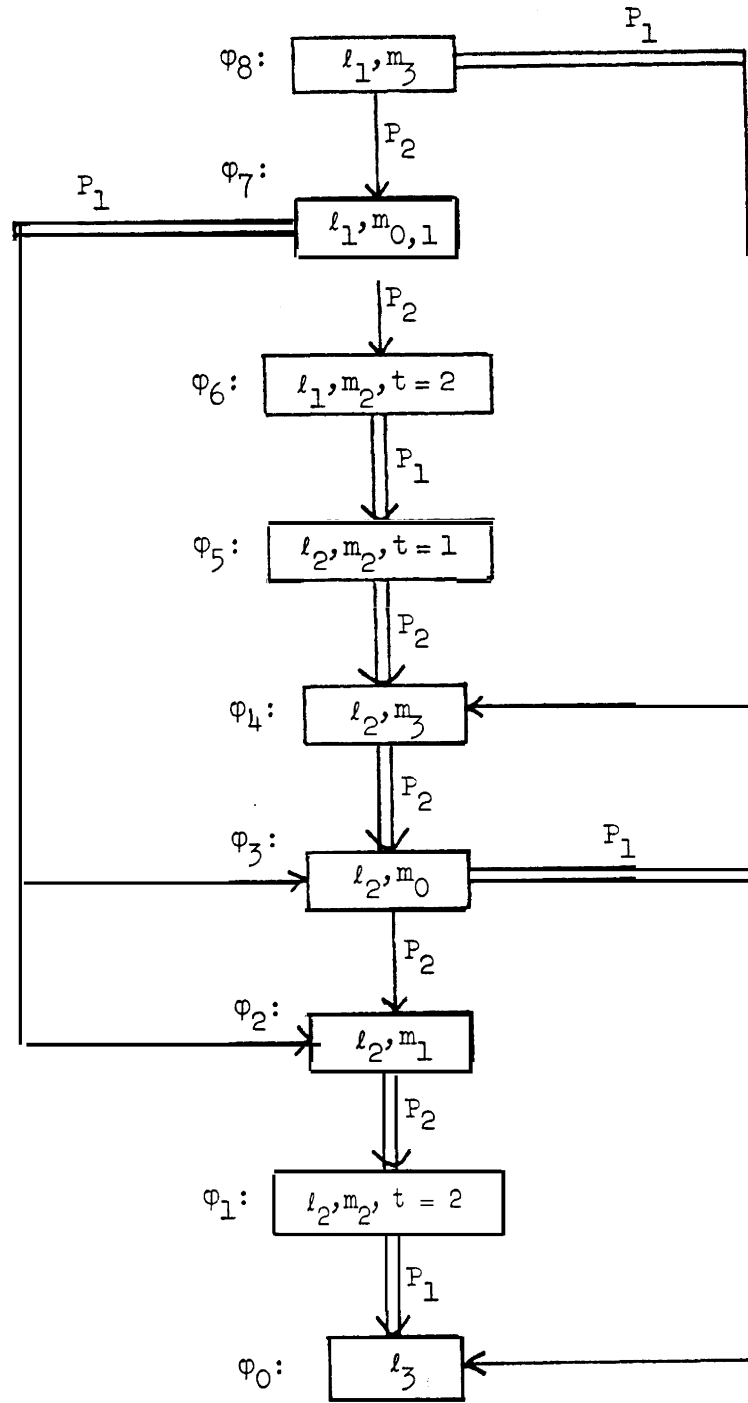
33

Fig. 2.    Proof Diagram for 2-bounded overtaking from $\ell_1$
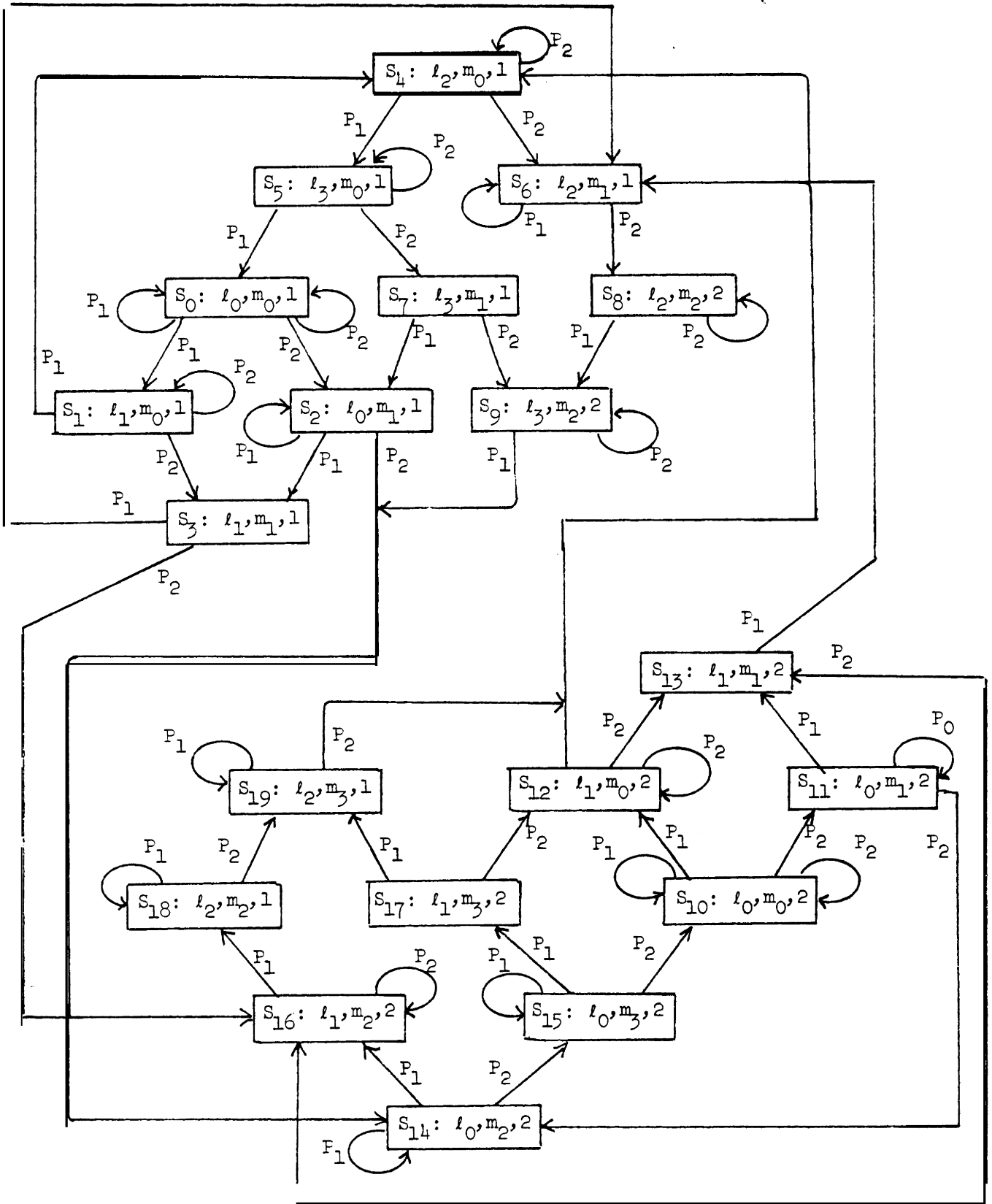
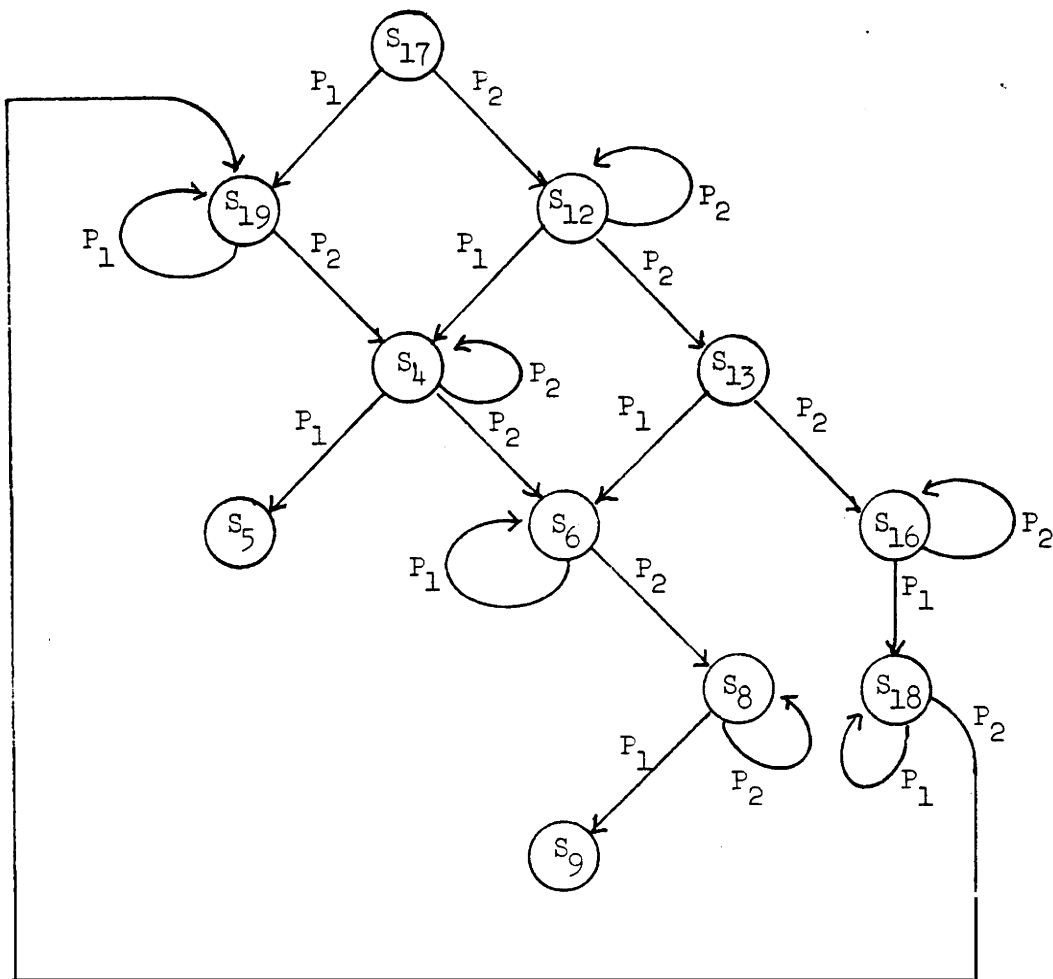Fig. 3. Joint Transition Diagram for the Mutual Exclusion Program,

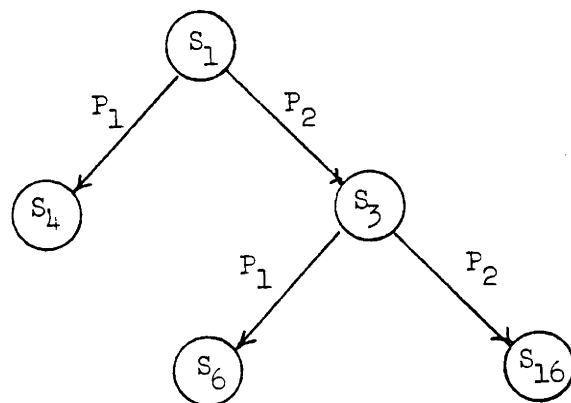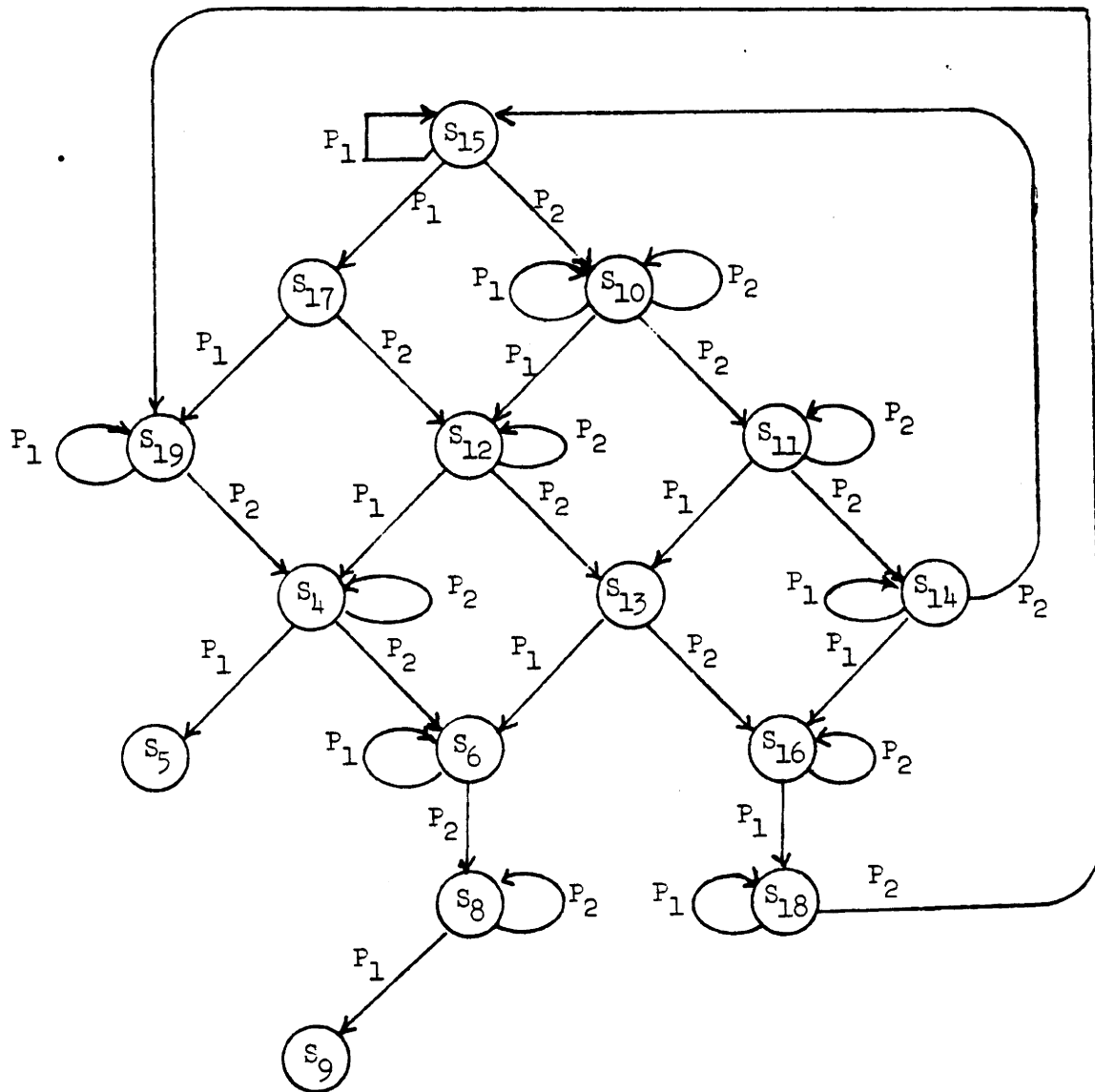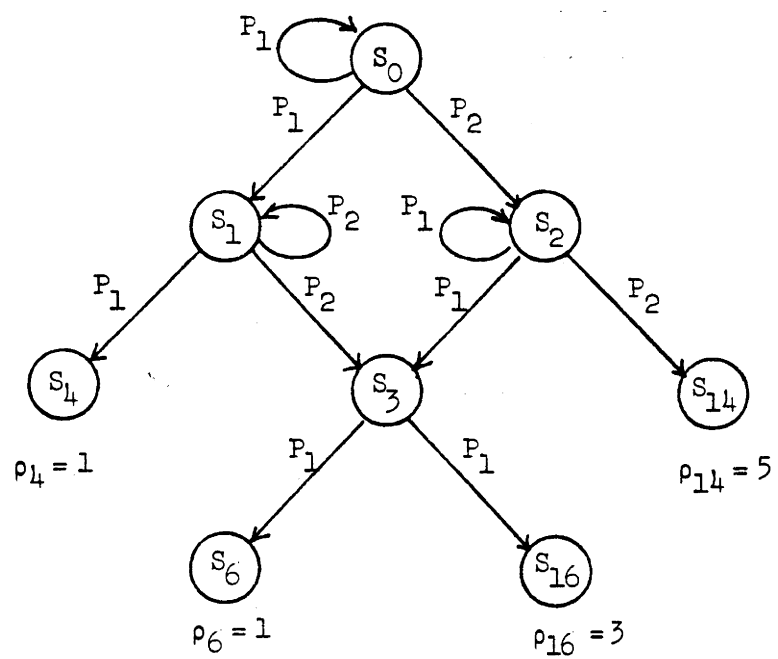Fig. 4.    $T_P(S_{17}, \underline{at}\ \ell_3)$



Fig. 5.    $T_P(S_1, \varphi_2)$

36

Fig. 6. $T_P(S_{15}, \varphi_0)$

Fig. 7.  $T_P(S_0, \psi_2)$