TEMPORAL LOGIC AND COMPUTER SCIENCE

By

Dov M. Gabbay

Draft May 1985

Department of Computing,

Imperial College of Science and Technology,

180 Queen's Gate,

London SW8 2SD

Telephone: 589-5111

Telex: 261503

Notation

(1) The classical connectives are denoted by

$\sim, \wedge, \vee, \rightarrow, \forall$ and $\exists$.

(2) $(T, <, =)$ denotes a model of partial order, i.e. $<$ is a binary relation on T, usually irreflexive and transitive. In this book $(T, <, =)$ will be called a flow of time. t,s,x,y, $\in T$ denote elements or moments of the flow of time and the quantifiers $\forall$ and $\exists$ range over them.

(3) We also have notation for subsets of T. These could be $Q \subseteq T$, $T_i \subseteq T$ or $T_a \subseteq T$, depending on the context. Binary relations on T may also be used, e.g. $R \subseteq T^2$. Formulas in the predicate logic of $(T, <, =)$ are denoted by **A**, **B**. The symbols appearing in **B** are sometimes explicitly written. Thus for example if we write **B** $(t,x,<,=,Q_i,R)$ we mean that **B** is in the predicate language of T with quantifiers over points of T, with exactly t, x the free variables of **B** and $Q_i \subseteq T$ and $R \subseteq T^2$, are the only non-logical symbols.

(4) Propositional temporal logics use letters p,q,r for atomic propositions and use the same logical connectives $\sim, \wedge, \vee, \rightarrow$. Other new temporal connectives are especially introduced. Well known among them are Fq, Pq (one place), and $U(p,q)$, $S(p,q)$ (two place), or in general $\#(q_1, \ldots, q_m)$ for m-place connectives. Formulas are denoted by A(q,r), B(q,r), sometimes indicating which atoms or connectives appear in them.

The letters **a**, **b** are used as all purpose letters for special notation of formulas either in the language of tense logic or in the predicate language of $(T,<,=)$.

2-4

(5) We talk about (T,<,=) being a flow of time. Given a flow and a temporal language, a temporal model (a temporal structure, or a model) is denoted by (T,<,=,h) and is obtained in conjunction with an assignment h. An assignment h is a function giving each atom a subset h(q)$\subseteq$T. h can also be described as a function h(t,q) giving {0,1} as values. Its meaning is intended as indicating at which moments t the atom q is true. Thus t $\in$ h(q) or h(t,q) = 1 reads "q is true at t". There may be variations on the above; we may have functions h(t,T1,q) depending also on T1 $\subseteq$ T. The exact role of h will be described in each context via agreed recursive procedures. These procedures are characteristic to the specific logic discussed.

(6) Val(A,h,t0,t1, ... , tm) or equivalently ||A||h,t0, ... , tm, σ denotes the truth value of the formula A under assignment h, at the time to with reference to points t1, ... , tm.

(7) The words tense logics are used when temporal logic is applied to the analysis of natural language.

PART 1: HOW TO PRESENT A TEMPORAL LOGIC

Chapter 1. The handling of time phenomena, introduction and survey:

1. Introduction

Time is involved in every aspect of human activity. As computer science is being applied in wider and wider areas of our lives it is essential to develop logical systems which can describe time dependent phenomena. There are at least five main areas of computer science where time is involved in an essential way and where temporal logic is used. These are:

1. Databases:

   Their management, updating, their use of knowledge representation involving time and logical deduction involving time dependent data.

2. Program development, specification and verification for concurrent programs and processes.

3. Hardware:

   VLSI, reasoning about circuits, object oriented processes.

4. Natural language processing.

5. Distributed systems. Protocols for sharing knowledge.

Let us describe in a bit more detail what is involved in each of these topics. Of course we will be able to say more once we are more familiar with systems of temporal logic.

1. Databases

Time is involved in three ways.

(a) Temporal logic is needed to describe a theory of updating and maintaining of databases. This requires the capability of talking about the logic of the updating and the evolution of databases in time. For example, we may have time dependent updating

rules or some global constraints involving time.

(b) Putting in the databases information which is time dependent requires a choice of a good temporal language. How to represent time dependent knowledge, how to access it and how to reason with it.

(c) The real world is a combination of the two types of dependencies above. We need to develop a logic of actions in time. These notions are of key importance to our new programming language.

2. Program specification and verification

We need to develop a temporal logic which can talk about properties of programs and the way they change states with execution.

In concurrent programs, the knowledge of the input output relations of each participant does not yield the knowledge of input output relation of the parallel execution. We need a logic we can use to prove from the initial description the correct behaviour of the program.

3. VLSI

One needs asynchronous timing. We need a temporal logic specifying the behaviour of pieces of hardware and possibly prove theorems on how to put systems together and maintain certain properties.

4. Natural language processing

Time is involved in an indirect way. We need to analyze the uses of the tense and aspect of the language in order to prepare it (interface) for the computer. This is besides the theoretical value of studying logical structure of time use in language.

5. The need of temporal logic for distributed systems is a new application. One describes the partial knowledge different agents have as dependent of present and future states of the world.

We shall later give more detail about these areas. Let us now turn to temporal logic.

The following are the main present day research areas of temporal logic.

1. Philosophical applications.

   Temporal logic is used in philosophy to clarify various concepts which have been studied since the time of Aristotle. Some examples are causality, historical necessity, identity through time, the notions of events and actions, etc.

2. Temporal logic application in computer science as described above.

3. Natural language. Logical analysis of the use of tense and aspect in natural languages. Logical time models for natural language.

4. Pure logical study of temporal logic within the framework of logic itself. Special topics here include:

   (a) Axiom system, theorem proving and proof theory. Decidability. Model theory.

   (b) Expressive power of temporal languages.

   (c) Applications of temporal logic to the pure theory of other logics (e.g. provability as a modal logic etc.)

   (d) Deductive reasoning involving time.

   To computer science all the above four aspects of the pure logical theory are of great importance.

Temporal logics can be presented in four different ways:

(a) Use predicate logic with an additional parameter for time.

(b) Use special temporal logics to express temporal phenomena.

There are two methods of presentation here.

(b1) Semantical presentation.

(b2) Presentation using axiomatic systems for the connectives.

(c) The forth method is via ~~a calculus of~~ events.

2. HOW TO PRESENT A TEMPORAL LOGIC, IN THE PREDICATE CALCULUS

The first method of accounting for temporal phenomena is to do it within the framework of the classical predicate calculus.

The idea is to add an extra variable for time to the language of predicate logic and make all predicates time dependent.

Thus if L (x, y) represents x loves y, we can say in predicate logic statements like John loves all those who do not love themselves by:

$$\forall x\,[\sim L(x, x, t) \rightarrow L(J, x, t)]$$

The parameter t appears everywhere.

One can now quantify over t and say time dependent properties, like

$$\forall t > \text{now}\ (L(m, j, t)).$$

Mary will always love John.

This approach is favoured by many because of our direct control over time points. We can quantify over t,s, compare them, even use skolem functions involving them. Here is another example.

John is taking a pill today and will take one every two days!

$$\text{Take}(J, \text{pill}, \text{now}) \wedge (\forall t, s > \text{today})$$

$$(\text{Take}(J,p,t) \wedge (t + 2 = s) \rightarrow \text{Take}(J, \text{pill}, s).$$

There are some problems about identity through time etc., but the computer scientists leave these problems to the delights of the philosopher.

However, even for simple sentences like the above, this approach is not ideal.

(i) The first "problem" is that the "logic" of the time is hidden. The temporal logic as described above is not predicate logic. The parameter t ranges over a special flow of time e.g. t = 0, 1, 2 ..., and this makes the logic special. For example, we

are committed to all the axioms involving the time parameter t dictated by the properties of the flow of time chosen. We shall discuss this point in chapter 4, when the formal definitions of how to present a temporal system predicate logic are given and discussed mathematically.

(ii) The second problem with using predicate logic as a temporal logic, is more fundamental, and has to do with the representation of time dependent data. Consider a simple example:

Since Maggie became Prime Minister the pound has steadily been going down and in fact it will continue to go down for as long as she remains Prime Minister.

This sentence is crystal clear to us and is understood immediately without any mental effort. Let us write it in predicate logic with time parameters.

Let PM (x, t) read: x is a Prime Minister, at time t

VP (x, t) read: the value of pound at t is x

n = now

m = maggie

The sentence becomes:

$$\exists t < n\,[PM(m,t)\ \exists s < t\ \forall u\,(s < u < t \rightarrow \sim PM(m,u))$$

$$\wedge \forall u\,(t < u < n \rightarrow PM(m,u)) \wedge$$

$$\forall u,v,x,y\,(t < u < v \leq n \wedge VP(x,u) \wedge VP(y,v) \rightarrow x < y)) \wedge PM(m,n) \wedge$$

$$\forall s > n\,[\forall u \leq s\ PM(m,u) \rightarrow \forall u, v\,(n \leq u \leq v \leq s \wedge VP(x,u) \wedge VP(y,v) \rightarrow x < y)]$$

The above sentence of predicate logic is not readable.

A point always claimed in favour of predicate logic is the existence of rich highly developed theorem proving techniques for it.

Take for example resolution methods; to present the above as a "ready for Resolution"

disjunctive clause we have to skolemise and then write it in disjunction forms. The "meaning" will be lost forever.

Another awkward example is the following. To take a mortgage on a house one needs an insurance policy stating that if the owner dies, the mortgage will be paid. Thus the conditions required by the bank are:

x gets a loan at time t if house of x is in good value and condition at time t and x financial position is acceptable at time t and x takes an insurance policy at timet.

How do we express the insurance policy clause in predicate logic? Our first attempt is to use a time parameter, after all, the insurance is a committment to pay in the future. Thus we write, x takes an insurance policy (of the above kind) at time t with company y iff for all $s \geq t$ (if x dies at s then y pays at s).

The above is not sufficient, because we can say in English:

John took an insurance policy but when he died, the company did not pay.

To express the above property we need for each moment of time t, to outline the planned future and the real future separately. It becomes awkward to do so in predicate logic.

The above objections are qualitative and stylistic but not mathematical. It is like saying that Pascal is nicer to write in than Basic but both languages are equivalent in mathematical power.

Can pure temporal logics such as e.g. axiomatic systems, say things which the predicate calculus cannot say at all?

This will be examined in later chapters.

For the time being note that the notion of expressiveness from the point of view of computer science is different from the logical notion. For example the finiteness of time is not a first order property but computationally it is very convenient. We cannot give

more details here, we shall have to define the exact notion of "A temporal logic presented within the framework of classical logic", we shall have to wait for chapter 4.

3. SEMANTICAL PRESENTATION OF TEMPORAL CONNECTIVES:

The second way of presenting a temporal logic is via connectives. We imagine a flow of time (T, <, now) and the various predicates e.g. John loves Mary and Maggie is P.M. get values T or F at each moment of time. So far it is equivalent to writing L (J, M, t) and PM (m, t), except that the t is suppressed.

The difference comes when we want to talk about the distribution of truth values through time. We allocate to the logic connectives which describe behaviour through time using the suppressed now as a reference point. This is very similar to the way we use connectives in English.

Example E1:

Consider the following, very basic connectives:

GA = A will always be true

HA = A was always be true

FA = A will sometimes be true

PA = A was sometimes be true.

S (A, B) = B was continuously true since A was true

U (A, B) = B will continuously be true until A is true

Thus for example:

John will love Mary becomes

FL (J, M)

meaning in predicate logic:

$\exists t > now\ L(J, M, t)$.

John has loved Maggie since Maggie was PM is:

S (PM (m), L (J, m))

in predicate logic we have to write:

$\exists t < now\ [PM(m, t) \wedge \forall s(t < s < now \rightarrow L(J,M,s))]$

Let us translate the sentence about the pound.

S (PM (m), PM (m) ∧ Pound going down)) ∧ PM (m)

∧ U (~ PM (m), PM (m) ∧ Pound going down)

We have to translate "Pound going down"

$\forall y\ \forall x\ [VP(x) \rightarrow S(PM(m), VP(y) \rightarrow y > x)]$

If the value of the pound now is x then since Maggie is PM if the value was y at any time then y > x.

This is not the best translation. We really should follow the way we perceive the notion "pound going down" and regard the sentence as having value at an interval.

The above translations are good no matter whether the temporal connectives are introduced via axioms or via a sematical interpretation.

We now describe the components needed to specify a temporal logic in a semantical way. These are:

(1a) The flow of time of the logic.

(1b) The units of time needed to give truth values.

(1c) Possible restrictions on the assignments to the atoms.

(2a) The temporal connectives used.

(2b) The truth conditions for the connectives.

The meaning of (1a) is clear. The flow of time can be the integers, the real numbers, partially ordered sets and so on.

(1b) chooses the units of time in which atomic sentences get truth values. These can be integers, intervals of integers, pairs of integers etc.

(1c) describe possible restrictions on the truth values given to the atoms. A typical example could be that each atom can be true in only a finite number of points (units of time).

(2) chooses the connectives and determines their meaning.

Example E2

We give five temporal logics, all based on the integer flow of time.

They differ either on (1b), on the unit of time for truth values or on (1c), the assignment to the atoms or on (2), the choice of connectives.

The logics are called (a), (b), (c), (d) and (e).

Example E2a

(a1) The unit of time for truth values is an integer.

(a2) The assignment to the atoms is arbitrary.

(a3) The connectives are (besides the classical ~, ∧, V, ->) Since (A, B) and until (A, B).

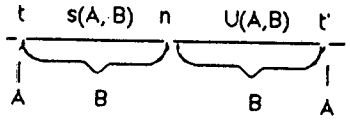
(a4) The truth table for the connectives is defined as follows:

An atom gets truth value at each moment of time.

S (A, B) is true at n if for some t < n, A is true at t and for all points between t and n, B is true.

U (A, B) is true at n if for some t' > n, A is true at t and for all points between n and t', B is true.

In a diagram:



FA and PA of the previous example can be defined as U (A, A → A) and

S (A, A → A) respectively.

Example E2b

(b1) Time is the integers.

The unit of time for truth values is an interval [m, n], m ≤ n.

(b2) There is no restriction on the assignment to the atoms.

(b3) The connectives are Next (A) and (A + B).

(b4) An atom is true or false at an interval. Thus we can have

q true at [1, 3]

q false at [1, 2]

q true at [2, 3]

We have no persistence criteria or any principles for subintervals. The truth tables for the connectives are:

Next A is true at [m, n], m ≤ n if m+1 ≤ n and A is true at [m+1, n].

(A + B) is true at [m, n] if for some k, m ≤ k ≤ n, we have, A is true at [m, k] and B is true at [k, n].

Example E2c

(c1) Time is the integers. The unit of time for truth values is a pair (m, n) of integers.

The connectives are F* and P*.

(c2) In this logic an atom is given truth values at points. The pair (m, n) is considered as an evaluation point m and a reference point n. Thus the truth value fo an atom q at (m, n) depends on m only.

The truth value of an arbitrary A may depend on both (m, n). Here is the table for F*A.

F*A is true at (m, n) iff the following cases hold:

m = n and for some k > m, A is true at (k, n)

m > n and A is true at (m, m)

m < n and for all k such that m < k < n, A is true at (k, k).

P*A has the symmetrical truth table.

P*A is true at (m, n) iff

m=n and for some k < n, A is true at (k, n)

m < n and A is true at (m, m)

m > n and for all k such that m > k > n, A is true at (k, k).

Example E2d:

The fourth logic, (d) is exactly like logic (a) with the added restriction on the assignment to atoms that the set of points in which the atom is true is either finite or its complement is finite.

Example E2e:

The fifth logic, (e) is the logic of execution sequences of a program.

(e1) Let the set of units of time be all infinitely proceeding sequences of integers, of the form **s** = (x0, x1, x2, ...).

Let $\mathbf{s}^n$ = (xn, x (n+1), ...) be the tail of **s**. Atomic sentences q get a truth value at each **s**.

(e2) We define the temporal connectives □q, ◇q, U (p, q) and Oq.

□q is true at **s** iff ∀n ≥ 0 (q is true at **s** n).

◇ q is true at s iff for some n > 0, q is true at s n.

0q is true at **s** iff q is true at **s**1.

□ corresponds to our familiar G, ◇ corresponds to F and 0 corresponds to next. Note however that the future in this system includes now as well. U (p, q) is true at s if for some n, p is true at sn and for all m ≤ n, q is true at **s** m.

Bold

4 AXIMATIC PRESENTATION OF TEMPORAL LOGIC

The axiomatic presentation follows the tradition of defining the meaning of the logical connectives using proof rules and deductive axioms.

Our first aim in this section is to give some examples and try and show that axioms on temporal connectives are very expressive. In fact, it seems that even the simplest set of connectives, namely {F, G, H, P} which are really nothing more than the existential "quantifiers", enable us to write axiom systems which can distinguish between different properties of the flow of time.

The weakest possible propositional temporal logic is the logic Kt. Its axioms are:

1. Axioms for classical logic (Take all tautologies as axioms).

   $G(A \wedge B) \leftrightarrow GA \wedge GB$

   $H(A \wedge B) \leftrightarrow HA \wedge HB$

   $GA \rightarrow GGA$

   $HA \rightarrow HHA$

   $\vdash A \Rightarrow \vdash GA$ and $\vdash HA$

   $A \rightarrow GPA$

   $A \rightarrow HFA$

The last two axioms say that the present moment is always in the past of any future moment and always in the future of any past moment.

This logic is complete for general partially ordered flow of time.

In general, there is a good correspondence between semantic conditions and axioms, with few exceptions. Different conditions on (T, <) can validate or invalidate wffs. Temporal logic wffs can be false in one type of flow of time and

hold true in others. There are wffs that are true in dense time, discrete time, or even complete time. Some characterise branching and so on.

Consider Hf (where f is falsity, e.g. g ∧ ~ g). We have:

||Hf||h, t = 1 iff ~ ∃s [s<t]

Thus in a flow of time satisfying ∀t ∃s [s<t], P H f V H f V F H f is always false. Otherwise it is true. This shows that the truth of wffs depend on the flow of time. In fact, some properties of the flow of time may be characterized by wffs. Take for example U (~f, f) S (~f, f). This is true at a point t iff t is discrete. Discreteness can also be expressed using P, F only.

Let us summarize the correspondence between conditions and wffs expressing these conditions: We will try to use P, F only.

(a) The condition that the present moment is in the past of any future moment corresponds to A → GPA, or to A → HFA. This condition is incorporated already in the notion of flow of time since when we write t<s we say t is in the past of s and also s is in the future of t. Symbolically t<s is the same as s>t.

(b) The transitivity of < corresponds to GA → GGA or equivalently to HA → HHA.

(c) The existence of an endpoint above any moment corresponds to Wff FGf.

(d) The condition that there exists one future doomsday common to all futures corresponds to

FGf ∧ F [F (A ∧ Gf) ∧ G (B ∧ Gf) → F (A ∧ B ∧ Gf)]

(e) The condition of linearity in the past corresponds to

PA ∧ PB → P (A ∧ B) ∨ P (A ∧ PB) ∨ P (PA ∧ B)

Total linearity is obtained by adding the Wff for linearity in the future.

(f) The condition of existence of tomorrow point corresponds to

F (GA ∧ A ∧ ~B) ∧ F (GB ∧ ~A) → F (F (GA ∧ A ∧ ~B) ∧ F (GB ∧ ~A)).

We saw before that U (~f,f) corresponds to this condition.

However, U is not definable using F, P but F is definable using U - (FA = U (A, A → A)).

(g) The condition ∀x ∀y (x < y → ∃u (x < u < y) is a condition of total density. We can take the opposite of discreteness for a point, namely ∀ s>t ∃u (t < u < x).

The corresponding Wff is FA → FFA.

(h) The condition of the well foundedness in the past corresponds to

P ~ A → P (~A ∧ HA)

(k) The condition of completeness (no gaps) corresponds to the Wff:

FGA ∧ F ~ A ∧ G (~A → F ~ A) → F (GA ∧ ~ PGA).

It is doubtful whether the condition can be expressed using F only. It can be expressed using U only as

FGA ∧ F ~ A ∧ G (~ A → F ~ A) → U (A ∧ GA, F ~ A)

Recall that FA = U (A, ⊤ → A)

and GA = ~F ~A.

Let us summarise this surprising correspondence between additional axioms and properties of flows of time. Here is a table

TABLE 1

Properties of time	Axiom
Irreflexivity (~ t < t)	no axiom,Cannot be characterized by axioms. Only by a rule.
Reflexity x < x	$GA \rightarrow A$ $HA \rightarrow A$
Linearity	$FA \wedge FB \rightarrow$ $F(A \wedge B) \vee F(A \wedge FB)$ $\vee F(FA \wedge B)$ Similarly for P.
Density of time	$FA \rightarrow FFA$ $PA \rightarrow PPA$
Time is dedekind complete (like the real numbers)	$L(GA \rightarrow PGA) \wedge GA$ $\rightarrow HA$ Where $LA \equiv A \wedge GA \wedge HA$
Time is finite	$FA \rightarrow F(A \wedge G \sim A)$ $PA \rightarrow P(A \wedge H \sim A)$.
Time is infinite	GF (truth) HP (truth)
Time is integers	as shown above
Some second order conditions on time	some corresponding axioms.

For example the axioms for Kt together with the axioms for linearity, density and infinity comprise an axiom system for rational time flow.

We see that axioms can express properties of time which first order logic cannot express. Thus the presentation of a temporal logic via axioms and rules can be sometimes more powerful than a semantical presentation.

On the other hand, there are semantical presentations which have no known corresponding axiom systems, like some logics of Historical Necessity.

We saw in the table that irreflexivity cannot be characterised by an axiom. It can be characterised however by a rule of inference, namely:

$$\frac{\sim q \wedge Gq \wedge Hq \rightarrow A \quad ; \quad q \text{ not in } A.}{A}$$

The rule says that if it is a theorem that whenever time is irreflexive then A is true ($\sim q \wedge Gq \wedge Hq$ can hold at t only when $\sim t < t$) then A is true anyway ($\vdash A$), because we can always, make time irreflexive and always make $\sim q \wedge Gq \wedge Hq$ true without affecting A, since q is not in A.

Using the above irreflexivity rule, we can prove the following wholesale axiomatization lemma.

Lemma L1.

Assume given a set of connectives #1, ..., #m for a temporal logic over a flow of time $(T, <, =)$, such that these connectives have 1st order truth tables. Then more connectives #m+1, ..., #m+k can be effectively found and added such that the logic with entire set of connectives #1 ... #(m+k) can be effectively axiomatised.

Of course the irreflexivity rule is used in the axiomatization. The entire procedure of finding the new connectives and the axioms is effective and computable.

Example E1:

We can now axiomatise the logic with since and until for any flow of time.

Take the corresponding axioms for the same flow of time using P, F, G, H and the rule for irreflexivity and add the axioms

$$Hq \wedge \sim q \wedge Gq \rightarrow [U(A, B) \leftrightarrow F(A \wedge H(B \vee Hq))]$$

$$Hq \wedge \sim q \wedge Gq \rightarrow [S(A, B) \leftrightarrow P(A \wedge H(B \vee Gq))]$$

What we are doing in these axioms is to effectively state the truth conditions fo U and S.

Example E2:

To give another example take the connective Nq reading: q is true around now, namely:

Nq is true at t iff $\exists x, y\ (x<t<y \wedge \forall z\ (x<z<y \rightarrow q \text{ is true at } z))$

The properties of N together with F and P can be characterised, without irreflexivity, by the following axioms, say for linear rational time.

(1) Axioms for F and P for linear rational time, (see table above).

(2) $q \wedge Hq \wedge Gq \rightarrow Nq$

$Nq \rightarrow NNq$

$Nq \rightarrow q$

$N(p \wedge q) \leftrightarrow Np \wedge Nq$

$G \sim q \rightarrow \sim Nq$

$H \sim q \rightarrow \sim Nq.$

Using irreflexivity we can write the following axioms for N.

(1*) Axioms for P, F for linear rational time including the irreflexivity rule.

(2*) $r \wedge H \sim r \wedge G \sim r \rightarrow$

$[Nq \leftrightarrow (q \wedge F(HPr \rightarrow q) \wedge P(GFr \rightarrow q))]$

Notice that the wff involving q in (2*) uses r as a parameter name for the point t and states the table of N. In the same manner we can axiomatise any connective. We mentioned that sometimes we cannot axiomatise a connective directly but with the help of other connectives, called intermediate connectives. The table of a connective # may be so complex that some intermediate connectives #i are

needed and an axiom system is written for {#, #i} together.

Using irreflexivity we can also axiomatise easily various properties of the flow of time. For example let us write an axiom saying each point has an immediate successor:

$q \wedge H \sim q \wedge G \sim q \rightarrow F$ (HPq → falsity)

There will be more discussion, of course, in the chapters dealing with axiomatizations.

Example E3:

To give further examples. Here is an axiom system for the temporal logic of Manna and Pnueli for program specification and verification for a shared variable concurrent system. This system was represented as example E2(e) of section 3.

Time is the integers. We deal with future only. The connectives are

$\Box A$ = A will always be true (beginning now)

$\Diamond A$ = A will sometimes be true including now

OA = A is true tomorrow

U (A, B) = B begins now to be true until A is true.

Axioms

$\sim \Diamond A \leftrightarrow \Box \sim A$

$\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$

$\Box A \rightarrow A$

$O \sim A \leftrightarrow \sim OA$

$O(A \rightarrow B) \rightarrow (OA \rightarrow OB)$

$\Box A \rightarrow OA$

$\Box A \rightarrow O \Box A$

$\Box(A \rightarrow OA) \wedge A \rightarrow \Box A$

$U(A, B) \leftrightarrow A \vee (B \wedge OU(A, B))$

$U(A, B) \rightarrow \Diamond A$

The inference rules are modus ponens and $\frac{A}{\Box A}$ and of course all tautologies are valid rules.

To the above we add quantifier axioms, to get the predicate temporal logic. If we add to the above system the following groups of axioms:

(1) Axioms about the program application domain.

(2) Axioms about the concurrent programming principles.

We get a logic which can be applied in program verification.

We can thus, for a given program, describe what it is and state the correctness lemma for it. We can then try and prove the correctness statement using the above axioms.

Let us give another example:

Example E4

Consider a propositional language with the connectives H and P (P being $\sim H \sim$) and the classical connectives $\wedge$, V, ~, $\rightarrow$.

The intended interpretation of H A is A has always been true and the flow of time we have in mind is finite past linear chain

-n, ... , -3, -2, -1, 0

we define the system Y1.

(1) The connectives are H,$\wedge$,v, ~, $\rightarrow$ (P is defined as ~ H ~)

(2) The language is propositional. Thus a wff is any atom q, and if A and B are wffs so are A $\wedge$ B, A V B, A $\rightarrow$ B, ~ A and HA and PA.

(3) A flow of time for Y1 is any finite chain of the form

0
-1
-2
|
|
-n n $\geq$ 0.

0 is the present and -1, -2 ... -n are the past points.

(4) An assignment of truth values is any function h giving values T or F to any atom q and a point of time i.e. h (-m, q) $\in$ {T, F}.

(5) The interpretation of HA is:

HA true at -m iff A is true of all points -n $\leq$ -m.

(6) A wff A is said to be valid if it gets value T in any flow of time for any h.

(7) The following is an axiom system for Y1.

(a) $\vdash$A for any truth functional tautology A.

(b) $\vdash A \Rightarrow \vdash HA$

(c) $H(A \wedge B) \leftrightarrow HA \wedge HB$

(d) $PA \wedge PB \rightarrow P(A \wedge B) \vee P(A \wedge PB) \vee P(B \wedge PA)$.
(axiom for linearity).

(e) $PA \rightarrow P(A \wedge H \sim A)$.
(axiom for finiteness).

The completeness theorem for Y1:

Y1 ⊢A iff A is valid (in every finite linear time model).

There are several methods for proving completeness, we mention two here.

The first method is a semantic tableaux method. We notice that axioms (e) says that if B was true there is a first time in which B was true and axiom (d) says that all past truths are linearly ordered. Thus given a wff A for which we want to find a model, we look at all subformulas Bi of A and if they were true in the past, we can order linearly the first time they were true. This gives us the tableaux model.

The second method is via complete theories and is more general. It will be given later, in chapter 3.

5. **Calculi of events**

Events arise mainly in connection with database updating and maintainance.

Databases are constantly changing. Information keeps coming in. The database is updated and modified. Consistency and integrity rules may need be maintained.

This requires a special language and logic which can handle change and dependence on time.

We need to develop a temporal logic for representing time dependent data and time dependent rules. This language and logic must have a good power of expression, must represent adequately our intuitive perception of dependence on time and change, and must properly interact and integrate with other components of our system (i.e. what we are discussing in other sections).

What are the major problems associated with our task?

A: Representation of time dependent data

The first problem is that of choosing the correct representation language for time dependent databases and rules. This is a serious problem. Many system rules work directly on data representation and so we must choose correctly.

Here are three extreme options available to us, which we will illustrate via an example.

Consider a rule A which is valid only during the period of time $1970 \leq t \leq 1980$. The rule A itself is not time dependent. For example, A may say that a person can teach mathematics in school if he has a degree in mathematics from Oxford or Cambridge. In 1970, this rule was introduced and in 1980, it was changed. A degree in education was also required, after 1980.

The problem is how to represent this rule. One way is to write assertion (a) below, where x stands for persons and t for a year:

(a) Teach math (x) if Oxbridge (x)

with a provision that this rule is valid during the period $1970 \leq t \leq 1980$. This corresponds more or less to the method of using connectives. Another way, is to put time explicitly in predicates and write equation (b) below: (corresponding to using predicate logic with variables for time):

(b) Teach math (x, t) if Oxbridge (x, t) and $1970 \leq t$ and $t \leq 1980$.

Expression (b) is not natural for this case. We may have an entire block of rules, all valid in the period 1970-1980, and the natural way to represent these rules is as in the diagram:

Rule	Restriction
(not mentioning time)	use only in the period 1970-1980

This is how we visualise this block of rules. It is also easy to ask from this representation which rules are valid in the period 1970-1980. If the rules are written in form (b), the answer to the latter question is not immediate.

There are advantages in the representation of form (b), in the case that the dependence on time is not by neat large blocks of rules, but through time interdependent rules like the following example:

(c) Q (x, t) if R (x, t + 5) and S (x, t - 3) and (t < 1991)

An example in English of such time dependence is a rule of the form:

(d) One cannot take a holiday in Spain three years in a row.

In natural language we find both types of time representation available. The emphasis, however, is not to refer to time explicitly: We use words like since, until, before, after, during, which give relative relationships between events.

For databases, we have to choose the right combination and compromise between the two representations.

There is a third way of representing time dependent data in a database, using events. We do not mention time points (like 1970, 1980) at all. We use the events themselves as the time reference points. We thus do not talk about 1970 or 1980 but rather make references like:

"When M. Thatcher was Prime Minster"

"Before the UK joined the EEC"

"When one could still be a mathematics teacher without a degree in education".

The database will contain the (probably partial) information of what events occurred before, during and after other events. This approach has its intuitive appeal. It does indeed contain features which are present in our everyday use of language. It may not be the predominant feature, but it could be extremely useful, especially in connection with partial information.

Researchers in time logic as applied to the logical and grammatical analysis of natural language, do attach great importance to theories of events.

The possible difficulties of such an approach for database management may be in the integration of the time component of the system with other components (i.e. consistency, addition, deletion, non-monotonic rules, etc). In principle, however, a compromise of all three approaches to time descriptions can be construed, because on can describe rules valid only if certain events (i.e. "M. Thatcher is Prime Minister") occur, rather than rules valid at a certain time (i.e. 1970-1980).

The problem of integration can be reduced to that of control, (which must be done in logic!!) of the form:

"one can use rule A only when the answer to query? E is yes!"

Where E describes a certain event.

The above amounts to saying something like:

"the traffic restrictions below are valid when children are out of school".

The events approach is at present being studied by R. A. Kowalski - M. J. Sergot and others.

B: Reasoning with time dependent data

The following is a good example (based on Kowalski-Sergot) which we use to illustrate the interactions possible between non monotonic reasoning and time.

Take the system of registering visitors to Britannic House. When a person visits the building, the computer is informed of his entry. When that person leaves, the computer is informed of his leaving. For this type of time dependence, the best representation is form (a). We can describe schematically the presence of Mr. Smith at Britannic House on Monday as follows:

Hour

- 17 leave
- 14 enter
- 13 leave
- 09 enter

To answer the query:

(q) Was Mr. Smith in Britannic House at 15 hours?

We can use several possible methods of computation.

(1) We can go backwards in time until we meet leave, in which case we say, no. If we meet enter, we say, yes. If we meet neither we say, no.

(2) We can go forward in time until we meet leave in which case we say yes or meet enter and say no. If we meet neither we say no.

(3) One can try both directions, just to play safe.

The problem arises when something goes wrong, and the information in the database is deficient. Consider the following diagram, representing the information in the database.

Hour

- 17 leave
- 13 leave
- 09 enter

Routine consistency checks or integrity checks would probably detect the anomaly.

What the system will do to correct the above anomaly depends on the non monotonic rules for the Britannic House system. If it is customary and routine to register leave for everyone at 17 hours, we may ignore the 17 hours leave for Mr. Smith and decide that Mr. Smith left at 13 hours.

If registration is very strict and done by hand, then we may decide that an enter is missing and put in the extra missing enter, with a time t marked unspecified. We thus will have the diagram below in the database.

Hour

- 17 leave
- t (unspecified) enter
- 13 leave
- 09 enter

Let us follow this latter possibility. Choosing the representation above can cause trouble, if we do not coordinate our time language with all the other components of the system. Depending on the way the computer conducts the search for the answer, we may have any of the following possible answers to our query (q).

(1) The answer is yes, because the search finds that Smith was in the building from 9 to 13 and from 13 to 15, except at 13 hours itself. The reason being that the

unspecified time t of entry is interpreted by the computer as near to 13 as necessary. This interpretation will have the undesirable effect that Smith was out of the building at 13 hours but immediately in the building half a second past 13 hours.

(2) The answer is no, because the computer will interpret the time t of entry as near to 17 as necessary. Thus we have that Smith was not in the building immediately before 17 but nevertheless left it at 17.

(3) The computer may not give an answer, or saying that it cannot answer, or has to delay, until the time t is specified. This has the undesirable effect that when asked about Smith's presence in the building at 1303, it will still not give an answer, even though a human would concede that it is not likely that Smith would actually register his leaving, just to come back after three minutes!

Another variation of (3) is that the computer, using negation as failure, will give the answer no to question (q) because t is not specified (and so it cannot succeed). However, when asked:

(q*) "Was Mr. Smith at Britannic House at any time between 13 hours and 17 hours"?

The computer will say yes (since an unspecified t is available). We thus have the undesirable situation that when asked about any specific time between 13 hours and 17 hours the computer will say no, but nevertheless the computer says yes to (q*).

In the theory of events approach, the situation may be less critical. We do not have time points but events. Thus the scale will be:

Hour

17 - Second leave

- Second enter

13 - First leave

09 - First enter

The fact that the second enter has no time attached to it is not so important, since the "second entry" itself is the time "point". The hours

(09, 13, 17) are just extra details.

We may not have here the problems outlined above, but other problems will arise. Unfortunately, Humans make use of time in a very complex way.

(4) This possibility is the worst of all. Our system may be such that, having found an answer to a query, it records the fact explicitly in order to save computation time, and not have to re-compute again what the same query is asked again or used in the future.

Thus approaching the query twice in different environments, the computer may obtain and record both a yes and a no answer.

Computer consistency maintenance rules may be activated, and a chain reaction may be set in motion, and all of this because of a simple oversight in registering the coming of a Mr. Smith!

C: Database updating

So far we discussed representation of time dependent data and rules in a time language. Time language is needed also for updating databases which describe static situations.

Imagine 100 boxes and 300 coloured bricks which can be distributed among the boxes. The database describes which brick is in what box. If we start moving the bricks around,

we have the problem of updating the database, in the most economical way. We need a time language to express from the outside global changes and constraints on the database. We may find it convenient to regard the operation of updating as a transition function from one database state to another. Although the particular updating operations are application specific, e.g. "move bick no. x to box no. y", the logic involved in manipulating the updating is universal.

We also have to assure that the time language used for time dependent data interacts conveniently with the updating language. Updating time dependent rules can be viewed from inside the database and from outside the database. These views must agree. For example we don't want to say from the outside "use this rule in 1970" and the rule to say "I am valid only in 1980". It is preferable to find a good common time language suitable for all time dependent manipulations of the database.

A time language is required to talk about the dependent databases and updating. There are several extreme possibilities, none of them completely satisfactory, great care must be taken into interaction with reasoning because potentially one can get into trouble.

We want to build a temporal language capable of describing adequately time dependent rules and database updating in an integrated way!

6. Expressive power of temporal connectives

We have seen how to specify a temporal logic semantically. We need a flow of time, a language with connectives and an evaluation function for atoms and connectives relative to the relevant unit of time.

The evaluation function need not be expressible in the 1st order theory of the flow of time. So conceivably we can have a connective saying: I am true at t if and only if the future is a well ordered set. As is well known, being well ordered is not a first order property.

It is interesting to look at temporal logics where the truth tables of the connectives are first order definable. Certainly in such cases we can hope to axiomatize the temporal logic in some way and if this logic is used for specification or correctness then we can hope for an automatic theorem prover to do the checking for us.

We now proceed to define more precisely what we mean by a 1st order temporal connectives of dimension n.

Let $(T, <, =)$ be a partially ordered set, called the flow of time. Consider the monadic first order theory of this flow of time. This means that we allow, besides $<$ and $=$ also unary predicates over time like $\underline{P}(t)$, $Q(s)$ and quantifiers over points. Thus we can write

B1 $(\underline{P}, Q, t) = \exists t' < t\,[\underline{P}(t') \wedge \forall s\,(t' < s < t \rightarrow Q(s))]$

This is a formula with the free variable t involving $\underline{P}$, Q.

Another formula could be

B2 $(\underline{P}, Q, t) = \exists t' > t\,[\underline{P}(t') \wedge \forall s\,(t < s < t' \rightarrow Q(s))]$.

A one-dimensional temporal logic with connectives #1, ... , #m where the connective #i has k (i) places, is defined by formulas **C1**, ... , **Cm** of the monadic predicate logic of time. **CI** has k(i) atomic monadic predicates and one free variable t. Thus #i (q1, ... , q

(k(i))) holds at t if **C**I (Q1, ..., Q (ki)), t) holds, where

Qj = {s | qj true at s}.

For example, **B1** above is the table for Since (p, q) and **B2** is the table for Until (p, q).

The above is just a formal definition of 1st order truth table.

A truth table for n-dimension will be a formula **C** (R1, ... , Rk, t1 ... tn) where R1, ... Rk are all m-place predicates and t1 ... tn are n free variables. Here we allow the use of full predicate logic with T, <, =, and allow for R1, ... , Rk to be m-place and not only monadic. Thus the table for a connective # (A1, ... , Ak) is defined by:

(q1, ... , qk) true at points t1 ... tn

iff **C** (R1, ... Rk, t1 ... tn) holds where Ri = {(s1, ..., sn) | qi is true at (s1, ... , sn)}.

So for example, the table for Next q is:

C1 (R, m, n) = m+1 < n $\wedge$ R (m+1, n).

The table for (q1 + q2) is

C2 (R1, R2, m, n) = $\exists k\ (m \leq k \leq n \wedge R1\ (m, k) \wedge R2\ (k, n))$.

Since we are using temporal logic to specify and prove properties in some application area, it is of vital importance to us to know the expressive power of temporal logics and be able to axiomatise the logic. We shall deal in this section with expressive power. We must know how to find expressive connectives and how to use them. We first explain what kind of problem we face.

Take the connectives Since and Until, for integer time. The wff which is the table of sine is **B1**. Thus S (A, B) is true at t means:

$\exists s < t\ \forall u\ [A\ (s) \wedge (s < u < t \rightarrow B\ (u))]$.

The expressions of the logic are built up form S, U by substitution. Thus we can write

U (S (S (A, B), ~ S (B, B), U (B, A)).

This is a complex expression. Its truth at a point t corresponds to a 1st order formula

D (A, B, t)

D is built up by the quantifiers of S and U which come in blocks of TWO namely ($\exists\ \forall$).

Problem 1:

Can we get all wffs **D'** (A, B, t) of the predicate logic of the time by substitution S, U within each other?

How about

$\forall s\ \exists t'\ \forall u\ (t < s \vee s > u \vee A\ (s) \vee B\ (u) \vee (A\ (t') \rightarrow A\ (u))]$

Is there a C (A, B) such that C (A, B) is built up from A, B Since and Until and is true at t iff the above wff holds?

We have three quantifiers here. Can we get an equivalent formula by S, U, (using blocks of two?)

The problem of expressive power of a temporal logic has several aspects:

(a) How expressive are (i.e. how many **D's** can one get from) the temporal connectives of our logic.

(b) Given a limited number of connectives, which we need for our area of application, how do we find what to add to make them more expressive?

(c) Can we find enough (but finite) number of connectives which give us the expressive power of all (full) quantification.

(d) How does this depend on the flow of time and the unit of time for evaluation?

Clearly for computer science application we would like some natural nice connectives which are useful and intuitive, very strong in expressive power and can be nicely axiomatised. For example for integer time, Since and Until are fully expressive. This means that for any formula **D** of predicate logic of the flow of time there is a

propositional formula E with **D** as its truth table i.e.

E is true at t iff **D** (t) holds.

Since and Until are not sufficient for rational number time. For example we cannot say using Since and Until that we have

```
        A      ~A
(---------|---------)
     gap      now
```

i.e. we have a sort of (A+ ~A) connective with a gap in the middle.

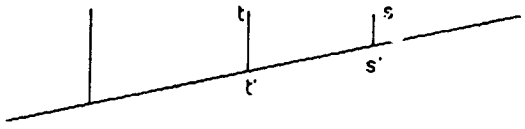
We need more connectives for rational time. Namely B until a gap with A true on the other side and ~B true on the other side of the gap, and similarly for "Since a gap".

The following concepts are related to expressive completeness

(a) Separation

(b) H-dimension

we explain Separation first. Suppose we have a flow of time and some connectives on it. Suppose we can, for any t, decompose the flow of time into a finite number of regions. Here is an example:



The obvious regions around t are:

1. points above t
2. t itself
3. points between t and t'
4. t' itself
5. points below t' and above to the side
6. points above t'

Suppose the connectives we have allow us to distinguish between the regions and the intersection of regions of t and regions of any s =/ t, (in our case we need 6 x 6 intersections).

Suppose further that the following holds.

Any wff of the logic can be rewritten in an equivalent way as a boolean combination of wffs dependent only on the regions (i.e. pure region wff).

Then we say the original set of connectives has the separation property over the flow of time.

The Separation Theorem

A set of connectives is expressively complete iff it has the Separation Property (plus some minor conditions).

Let us see what this means in terms of Since and Until and integer time.

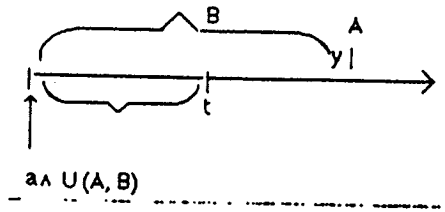
```
----------|----------
          t
```

The natural regions are Past, now, Future.

Separation means that every sentence involving Since and Until can be rewritten as a combination of pure past, present and pure future sentences.

Here is an example.

Consider D = S (a ∧ U (A, B), q). This is not a pure sentence. U (A, B) is embedded inside an S. Let us draw a diagram



There are three cases depending where y is: i.e.

(1) $y < t$

(2) $y = t$

(3) $y > t$

(1) If $y < t$ then $D = D1 = S(A \wedge q \wedge S(a, B), q)$

(2) If $y < t$ then $D = D2 = S(a, B \wedge q) \wedge A$

(3) If $y > t$ then $D = D3 = S(a, B \wedge q) \wedge B \wedge U(A, B)$.

These are the only cases thus

$D = D1 \vee D2 \vee D3$.

Thus D is a boolean combination of pure sentences.

In fact if we examine all possible cases of impure nesting of U within S i.e.

$S(a \wedge U, q)$

$S(a \wedge \sim U, q)$

$S(a, q \vee U')$

$S(a, q \vee \sim U')$

$S(a \wedge U, q \vee U)$

$S(a \wedge \sim U, q \vee \sim U)$ (same U in both

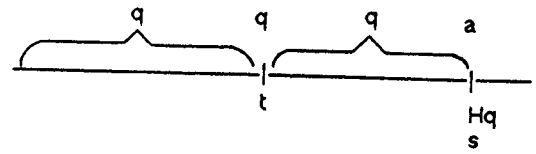
$S(a \wedge \sim U, q \vee U)$ sides) here

$S(a \wedge U, q \vee \sim U)$

and rewrite each case as a combination of pure wffs then this together with a careful induction argument will prove separation of S, U over the integers and show that our language has the same power of expression as quantifying over the integer time itself.

If we have a set of connectives and we don't know whether it is expressively complete, we can find what is needed by actually trying to separate and set how we get stuck.

Assume we have F, P, G, H and integer time. Take F (H q a)



at t, q is true, Hq is true, but q is true Until s1 we cannot express that and we find that we need Until. Thus:

$F(a \wedge Hq) = q \wedge Hq \wedge U(a, q)$.

In fact until cannot be defined using F, P, H, G.

7. H-DIMENSION:

The property of whether or not one can find at all a finite set of fully expressive connectives on a flow of time turns out to be a property of the flow of time itself.

Can we characterise this property by some other means, hopefully very simple and intuitive and then use it to show, for an application oriented flow of time, that full expressiveness is not available? The answer is yes and the property is H-dimension.

Let us begin with writing down that these exist at least two different elements. How do we write that? It is very simple. We write

$$\exists x\, y\, (x \neq y).$$

How about three different elements?

$$(\exists x\, y\, z)\, (x \neq y \wedge y \neq z \wedge x \neq z).$$

Now try to say the same thing without using three different variables letters. Use only two.

Is it possible to say that there exist 3 different elements using two letters only?

The answer is, generally not. Sometimes yes, depending on the flow of time. If time is linear, for example, we can write

$$\exists x\, [\exists y\, (y < x) \wedge \exists y\, (y > x)]$$

We used here only x, and y was reused. We know that the second y (y>x), is different from the first $\exists y\, (y < x)$.

If we did not have a total linear order we could not guarantee the existence of 3 elements because the above formula would have said more.

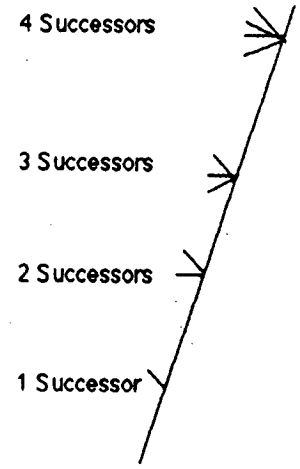
A flow of time has an H-dimension $\leq m$ if any sentence about the flow of time involving $<$, unary Predicates $P(t)$, $Q(t)$, free variables and quantifiers, can be equivalently written using at most m different bound variable letters.

Linear order can manage with 3.

H-dimension Theorem:

A flow of time admits a fully expressive finite set of temporal connectives if and only if (more or less) it has a finite H-dimension.

A corollary of this theorem is that the partially ordered set which allows for arbitrary large co-chains cannot have a finite set of fully expressive connectives, eg. the following tree cannot have a set of fully expressive connectives.



8. DECIDABILITY

Most temporal propositional systems are decidable. It is possible to cook up examples of undecidable ones.

Predicate temporal logics tend to be more complex than 1st order logics. For example the predicate logic with F,P,G,H over the real numbers is not even arithmetical.

There are two main methods of proving decidability. One is to express the semantics in a decidable theory, usually the monadic 1st order theory of W successor functions. The other is to show that the logic has the so called finite model property, namely that if A is not a theorem then A can be falsified in a finite model. Thus an axiomatizable logic with f.m.p is decidable, since we are going to have machines for generating both its theorems and non-theorems.

9. SAMPLE APPLICATION TO COMPUTING

We consider two examples here, two temporal systems which arose from application to computing. These are the systems described in example E2 of section 3. The system E2(e) of Manna and Pnueli, and the system E2(b), of Ben Moschowski.

Example E1

The Manna Pnueli system with Until, 0,□ and ◇.

This temporal logic can be used to talk about the exectution of a program by formalising the states and the transition rules of a program we can prove properties of the program. We base our use of temporal logic on the view of programs as generators of execution sequences. If we allow nondeterministic programs that each input to program generates a set of possible execution sequences of the form above. By stating properties which hold for all the execution sequences of a program, we are stating properties of the generating program.

To illustrate this point, let a program be labelled by lo ,..., le. lo being the entry point and le being the exit point.

Let atli be a predicate true at stage n if at that stage the program is about to execute li.

Partial correctness can thus be stated ((A,B) is the input output specification).

$atl_0 \wedge \bar{y} = x \wedge A(x) \rightarrow \Box(atl_e \rightarrow B(x, y))$

This reads:

If at any time the variable $\bar{y}$ equals input x and A (x) holds then whenever we reach termination point le B (x, y), where y is the current values of the program variable.

Total correctness is:

$atl_0 \wedge \bar{y} = x \wedge A(x) \rightarrow \Diamond(atl_e \wedge B(x, y))$.

◇ says we will terminate.

We can classify properties according to the form of the temporal formula.

Form $q \rightarrow \Diamond q$ is used for invariance properties such as safety, partial correctness, mutual exclusion, clear performance, deadlock freedom.

$q \rightarrow \Diamond q$ expresses eventualities, e.g. liveness total correctness, accessibility, livelock, freedom etc.

Here are some more examples of what this language can express.

a) Response to Insistence

The weakest form of responsiveness states that a permanent holding of a condition or a request p will eventually force a response q.

This can be written as: $\Box p \rightarrow \Diamond q$, or if stated over all future behaviours $\Box(\Box p \rightarrow \Diamond q)$.

Sometimes, the response q frees the requester from being frozen at the requesting state. In this case once q becomes true, p ceases to hold, apparently falsifying the hypothesis $\Box p$. This difficulty is only interpretational and we can write instead the logically equivalent condition $\sim \Box(p \wedge \sim q)$, namely, it is impossible for p to be frozen indefinitely and q never to realize.

To illustrate the utilization of such a statement, consider a component of a process which is busy waiting for a condition (say, $x > 0$) which presumably some other process is expected to generate.

m : if x = 0 then go to m

We use the proposition 'atm' which is true at each time instant in which the execution of our process is at m, namely about to execute m. Then , the only statement we can make

about this situation and its resolution is:

$\sim \Box$ (atm x >0)

i.e. it is impossible for the process to be stuck at m while x is continuously positive. It implies that the scheduler will eventually schedule this process which will find x positive and proceed beyond m.

This is also the weakest definition of a semaphore's behaviour, as well as the minimal fairness requirement from any scheduler. It requires that any process will eventually be scheduled for execution. Note that for semaphores this allows infinite overtaking.

b) Response to Persistence

A stronger requirement and the one which most suits a semaphore's behaviour is that the infinitely repeating occurrence of the condition p will eventually cause q. We do not require p to hold continuously, but only to be true infinitely often. The statement of this eventuality is:

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

or if required continuously:

Similarly to the case above the first form is logically equivalent to $\sim \Box(\Diamond q \wedge \sim q)$ which we will sometimes prefer.

Consider a semaphore instruction:

m : p(x)

The proper statement of its behaviour is:

$\sim \Box(x>0) \wedge$ atm)

This states that no process can be waiting indefinitely at m for a semaphore entry while the semaphore's variable turns true (positive) infinitely often. Note that to ensure this it

is not sufficient to guarantee that this process will be scheduled infinitely often. Because by some highly improbable run of bad luck all of these scheduling instances could exactly coincide with the instances in which x=0, and the process will never proceed. The same criterion should also apply to the fairness of conditional critical section instructions:

m : with r when B do S.

Here we should also require:

$\sim \Box(\Diamond (r>0 \wedge B) \wedge atm)$

i.e. it is impossible to remain stuck forever at m while states in which both B hold and the resource (r) is free, repeat infinitely often. This can be implemented by means of semaphores and a queue, or using an unbounded counter. It cannot be done with semaphores alone.

A weaker interpretation of the conditional critical section is also possible:

$\sim \Box (\Diamond (r>0) \wedge atm \wedge B)$

This one guarantees admission only if B is permanently held true from a certain point on, while r becomes free (positive) infinitely often. The implementation of this weaker construct by semaphores is easy.

c) Response to an Impulse

The strongest type of responsiveness is the one in which a single occurance of p guarantees q. This is written as: $p \rightarrow \Diamond q$ (more generally $\Box(p \rightarrow \Diamond q)$). It is a natural and useful expression for the discussion of total correctness, 'sometimes' reasonings, and other temporal causalities of internal events. It is too strong, on the other hand, for the expression of responses to external signals and conditions. This is so because it is

seldom the case that a system is so attentive that it could always detect a single occurance of an incoming signal which does not repeat.

Having formulated the different types of responsiveness, we can augment them with requirements for fairness. While managing quite well in cases a) - c) with just the $\Diamond$, $\Box$ operators, we must introduce now the 'Until' operator U.

Consider the following situations:

d) Absence of Unsolicited Responses

We may wish to complement the statement that p will eventually cause q ($p \rightarrow \Diamond q$), by saying that on the other hand, q will never happen unless preceded by a p. We may wish to state for example that a resource allocation system will not grant a resource to somebody who did not request it. Ignoring boundary effects (i.e. p and q happening simultaneously), this can be expressed as:

$\Diamond q \rightarrow U(p , \sim q)$

i.e. that is if q is going to happen at all, it cannot happen until p happened first (or concurrently).

e) Strict Fairness

Suppose there are two requests p_1 and p_2 and two corresponding responses q_1 and q_2. We may wish to impose strong FIFO discipline on the responding agent and state that if p_1 preceded p_2 then q_1 will precede q_2.

For this it is conventient to express the fact that starting from the present, the first occurance of p_2 must be preceded by an occurance of p_1. In fact this is exactly the expression used in d) above and we define generally:

$Pr(p_1, p_2) = \Diamond p_2 \rightarrow U\ (p_1, \sim p_2)$

The strict responsiveness discipline (FIFO) can now be written as:

$Pr(p_1, p_2) \rightarrow Pr(q_1, q_2)$

Consider for example two competing semaphore instructions:

m : P(x) n : P(x)

m': n':

A FIFO implementation of semaphores would require:

Pr (atm, atn) → Pr (atm', atn').

One could use this as a basic axiom for the behaviour of semaphores under a given FIFO) implementation and then deduce from it a global FIFO behaviour of larger parts of the program.

d) and e) are really only a first approximation to the properites we had in mind. For example Pr(p,q) strictly ensures only that the first occurance of q is preceded by an occurance of p. What exactly we would like to have is: "Every q is preceded by a p which happened after the last q, if any". Thus we cannot be satisfied with a single p succeeded by many q's, even though this situation formally satisfies the requirement: "every q is preceded by a p". The property described above i.e. interleaving of at least one p between consecutive q's can be described as:

$Pr(p,q) \wedge \Box (q \rightarrow Pr(p,q))$

It states that picking as a reference point either the initial state, or any instant in which q holds, ensures that the first instance of q after the reference point is preceded by an instance of p occuring between the reference point and the instance of q.

f) Mutual Exclusion

P1 and P2 are two processes wanting to enter a critical area. P1 controls the variable yi exclusively. The variable t is shared. P1 makes y1 true when interested in entering critical section and makes t = 1. P1 can enter if either y2 is false or t =1. It makes y1 = false when exiting. P2 performs the symmetrical actions when it wants to enter the area. We want to prove in logic that there is no deadlock i.e.

⊢P1 is waiting (i.e. y1 = t) → ◇(P1 is in critical section).

Here is the diagram of the sequence of events!

start: y1 = false

y2 = false

t = 1

l0 non critical	m0
l1 y1 =: true	m1 y2 =: true
l2 t =: 1	m2 t =: 2
l3 if y2 is false go to l5	m3 =: if y1 =: F go to m5
l4 if t =: 1 go to l3	m4 if t =: 2 go to m3
l5 critical section	; m5 critical section
l6 y1 =: false	y2 =: false m6
l7 go to l0	m7 go to m0

Example E2:

Let us now consider the internal time logic (b) of Example E2 of 3 and see what we can express using its connectives. Recall that evaluation is at intervals [m,n]. The connectives are Next A and (A + B).

(1) □A is true at [m,n] if for all $m \leq m' \leq n' \leq n$.

A is true at [m',n'].

This can be expressed as: ~ (true + ~ A + true).

(2) Let empty = ~ Next true.

Then it holds at [m,n] if m = n.

(3) Let skip = Next empty

It says that the interval has only one point.

[skip + skip + skip .... n times]

says the interval has n points.

Thus we can say:

(4) (skip + skip + q) =

q holds after two units of time.

(5) (skip ∧ q) = q is true at some unit subinterval.

(6) The yield operator $A \rightarrow\rightarrow B$ holds at [m,n] iff for any k, $m \leq k \leq n$, if A holds at (m,k) then B holds at [k,n].

$A \rightarrow\rightarrow B$ can be defined as ~ (A + ~ B).

(7) (true $\rightarrow\rightarrow$ A) says A is true at some final tail subinternational.

(8) $\langle i \rangle$ q = (q $\rightarrow\rightarrow$ true) : q true at some initial subinterval.

(9) Let beg q = (empty ∧ q $\rightarrow\rightarrow$ t) mean q begins

Let fin q = (true $\rightarrow\rightarrow$ empty q). It means that q is the final state

(10) Let = $\langle t \rangle$ q = (true $\rightarrow\rightarrow$ q). This reads q is true at some terminal subinterval. Let $[t]$ q read:

At all terminal subintervals q is true.

Then

(11) halt q = $[t]$ (q ´ empty).

It reads: the interval finishes only when q is true.

(12) Let keep q = $[t]$ (~ empty → q).

This reads that q is true in all non-empty terminal intervals.

This example is based on the logic of true intervals, called Tempora by B. Moschkowski, it allows for propositional quantification, which will be discussed in detail in a later chapter. Meanwhile, assuming we can quantify over propositions, we can define:

(13) q Until p = ∃x [beg x ∧ $[t]$ (beg x → [p ∨ (q ∧ Next beg x))]].

x is a new propositional variable, x is initially true and inductively remains so until q is true.

(14) We can iterate:

2q = q + q

nq = q + (n - 1) q

$$^{*}q = \sum_{1}^{\infty} q = q + q + q + \ldots\ldots$$

*q = def: ∃x (beg x ∧ $[t]$ (beg x → empty ∨ $\langle i \rangle$ (q Next halt beg X)))))

This says that the interval is of infinite length.

(15) While p do q = def.

[*((beg p) ∧ q) ∧ fin (~ p))