1

[illegible]

Stanford Artificial Intelligence Laboratory
Memo AIM-330

September 1979

Computer Science Department
Report No. STAN-CS-79-751

THE MODAL LOGIC OF PROGRAMS

by

Zohar Manna
Stanford University

and

Amir Pnueli
Tel-Aviv University

Research sponsored by

Advanced Research Projects Agency
and
National Science Foundation

COMPUTER SCIENCE DEPARTMENT
Stanford University

AD-A155 071

Stanford Artificial Intelligence Laboratory
Memo AIM-330

September 1979

Computer Science Department
Report No. STAN-CS-79-751

Accession For	
NTIS GRA&I	[✓]
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A/1	

THE MODAL LOGIC OF PROGRAMS

by

Zohar Manna
Stanford University

and

Amir Pnueli
Tel-Aviv University

ABSTRACT

We explore the general framework of Modal Logic and its applicability to program reasoning. We relate the basic concepts of Modal Logic to the programming environment: the concept of "world" corresponds to a program state, and the concept of "accessibility relation" corresponds to the relation of derivability between states during execution. Thus we adopt the Temporal interpretation of Modal Logic. The variety of program properties expressible within the modal formalism is demonstrated.

The first axiomatic system studied, the *sometime system*, is adequate for proving total correctness and 'eventuality' properties. However, it is inadequate for proving invariance properties. The stronger *nexttime system* obtained by adding the *next* operator is shown to be adequate for invariances as well.

This research was supported by the Advanced Research Projects Agency of the Department of Defense under ARPA Order No. 2494, Contract MDA903-76-C-0206 and National Science Foundation under Contract NSF MCS76-83655. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of Stanford University, or any agency of the U. S. Government.

THE MODAL LOGIC OF PROGRAMS

by

Zohar Manna
Stanford University and Weizmann Institute

and

Amir Pnueli
Tel-Aviv University

ABSTRACT

This document ~~We~~ explores the general framework of Modal Logic and its applicability to program reasoning. The authors ~~We~~ relate the basic concepts of Modal Logic to the programming environment: the concept of "world" corresponds to a program state, and the concept of "accessibility relation" corresponds to the relation of derivability between states during execution. ~~Thus we adopt the~~ Temporal interpretation of Modal Logic is adopted. The variety of program properties expressible within the modal formalism is demonstrated.

The first axiomatic system studied, the sometime system, is adequate for proving total correctness and 'eventuality' properties. However, it is inadequate for proving invariance properties. The stronger nexttime system obtained by adding the next operator is shown to be adequate for invariances as well. Additional keywords: computer logic.

This paper was presented as an invited address at the Sixth International Colloquium on Automata, Languages and Programming, Graz, Austria, July 1979.

This research was supported in part by the National Science Foundation under Grant MCS76-83655 and by the Advanced Research Projects Agency of the Department of Defense under Contract MDA903-76-C-0206.

I. THE GENERAL CONCEPTS OF MODAL LOGIC

In the hierarchic development of logic as a formalization tool, we can observe different levels of variability. Propositional Calculus was developed to express constant or absolute truth stating basic facts about the universe of discourse. This framework mainly deals with the question of how does the truth of a composite sentence depend on the truth of its constituents. In Predicate Calculus we deal with variable or relative truth by distinguishing the statement (the predicate) from its arguments. It is understood that the statement may be true or false adcording to the individuals it is applied to. Thus we may regard predicates as parameterized propositions. The Modal Calculus adds another dimension of variability to the description by predicates. If we contemplate a major transition in which not only individuals are changed, but possibly the complete structure of basic premises and meaning of predicates, then the Modal Calculus suggests a special notation to denote this major change. Thus any chain of reasoning which is valid on Earth may become invalid on Mars because some of the basic concepts naturally used on Earth may assume completely different meanings (or become meaningless) on Mars. Conceptually, this calls for a partition of the universe of discourse into worlds of similar structure. Variability within a world is handled by changing the arguments of predicates, while changes between worlds are expressed by the special modal formalism.

Consider for example the statement: "It rains today." Obviously, the truth of such a statement depends on at least two parameters: The date and location at which it is stated. Given a specific date t_o and location ℓ_o, the specific statement: "It rains at ℓ_o on t_o" has propositional character, i.e., it is fully specified and must either be true or false. We may also consider the fully variable predicate rain (ℓ,t) : "It rains at ℓ on t " which gives equal priority to both parameters. The modal approach distinguishes two levels of variability. In this example, we may choose time to be the major varying factor, and the universe to consist of worlds which are days. Within each day we consider the predicate rain(ℓ) which, given the date, depends only on the location. Alternately, one can choose the location to be the major parameter and regard the raining history of each location as a distinct world.

As is seen from this example, the transition from Predicate Logic to Modal Logic is not as pronounced as the transition from Propositional Logic to Predicate Logic. For one thing it is not absolutely essential. We could manage quite reasonably with our two parameter predicates. Secondly, the decision as to which parameter is chosen to be the major one may seem arbitrary. It is strongly influenced by our intuitive view of the situation.

In spite of these qualifications there are some obvious advantages in the introduction and use of modal formalisms. It allows an explicity discrimination of one parameter as being appreciably more significant than all the others, and makes the dependence on that parameter implicit. Nowadays, when increasing attention is paid to the clear correspondence between sumtax and natural reasoning (as is repeatedly stressed by the discipline of Structured Programming), it seems only appropriate to introduce extra structure into the description of varying situations. Thus a clear distinction is made between variation within a state, which we express using predicates and quantifiers, and variation from one state to another, which we express using the modal operators.

The general modal framework considers therefore a <u>universe</u> which consists of many similar <u>states</u> (or <u>worlds</u>) and a basic <u>accessibility relation</u>

between the states, $R(s,s')$, which specifies the possibility of getting from one state s into another state s'.

Consider again the example of the universe of rainy days. There, each state is a day. A possible accessibility relation might hold between two days s and s' if s' is in the future of s.

The main notational idea is to avoid any explicit mention of either the state parameter (date in our example) or of the accessibility relation. Instead we introduce two special operators which describe properties of states which are accessible from a given state in a universe.

The two modal operators introduced are $\Box$ (called the necessity operator) and $\Diamond$ (called the possibility operator). Their meaning is given by the following rules of interpretation, informally expressed, in which we denote by $|w|_s$ the truth value of the formula w in a state s

$$|\Box w|_s = \forall s'[R(s,s') \supset |w|_{s'}]$$

$$|\Diamond w|_s = \exists s'[R(s,s') \wedge |w|_{s'}] \quad .$$

Thus, $\Box w$ is true at a state s if the formula w is true at all states R-accessible from s. Similarly, $\Diamond w$ is true at a state s if w is true in at least one state R-accessible from s.

A modal formula is a formula constructed from proposition symbols, predicate symbols, function symbols, individual constants and individual variables, the classic logic operators (including equality) and quantifiers, and the modal operators. The truth value of a modal formula at a state in a universe is found by a repeated use of the rules above for the modal operators and evaluation of any classic (non-modal) subformula on the state itself. It is of course assumed that every state contains a full interpretation for all the predicates in the formula.

For example, the formula $\mathrm{rain}(\ell) \supset \Diamond \sim \mathrm{rain}(\ell)$ is interpreted in our model of rainy days as stating: For a given day and a given location ℓ, if it rains on that day at ℓ then there exists another day in the future on which it will not rain at ℓ; thus any rain will eventually stop. Similarly, $\mathrm{rain}(\ell) \supset \Box\, \mathrm{rain}(\ell)$ claims that if it rains on that day it will rain everafter. Note that any modal formula is always considered with respect to some fixed reference state, which may be chosen arbitrarily. In our example it has the meaning of 'today'.

Consider the general formula $\Diamond \sim w \equiv \sim \Box w$. As we can see from the definitions this claims that there exists an accessible state satisfying $\sim w$ if and only if it is not the case that all accessible states satisfy w. This formula is true in any state for any universe with an arbitrary R.

Given a more precise definition, a <u>universe</u> consists of a set of <u>states</u> (or <u>worlds</u>), on which a relation R, called <u>accessibility relation</u>, is defined. Each state provides a domain and a first-order interpretation over the domain to all the proposition symbols, predicate symbols, function symbols, individual constants, and individual variables in the vocabulary under consideration. A formula which is true in <u>all states</u> of every universe is called <u>valid</u>. Thus the above formula $\Diamond \sim w \equiv \sim \Box w$ is a valid formula.

Following is a list of some valid formulas:

A1*. $$\Diamond \sim w \equiv \sim \Box w\ .$$

This establishes the connection between "necessity" and "possibility".

A2*. $$\Box (w_1 \supset w_2) \supset (\Box w_1 \supset \Box w_2)\ ,$$

i.e., if in all accessible states $w_1 \supset w_2$ holds and also w_1 is true in all accessible states, then w_2 must also be true in all of these states.

The formulas A1* and A2* are valid for any accessibility relation. If we agree to place further general restrictions on the relation R , we obtain additional valid formulas which are true for any model with a restricted relation. According to the different restrictions we may impose on R we obtain different modal systems. In our discussion we stipulate that R is always <u>reflexive</u> and <u>transitive</u>.

A3*. $\Box w \supset w$ (equivalently $w \supset \Diamond w$) .

This formula is valid for any reflexive model. It claims for a state s that if all states accessible from s satisfy w , then w is satisfied by s itself. This is obvious since s is accessible from itself (by reflexivity).

A4*. $\Box w \supset \Box\Box w$ (equivalently $\Diamond\Diamond w \supset \Diamond w$) .

This formula is valid for transitive models. The equivalent form claims that if there exists an s_2 accessible from s_1 which is accessible from s such that s_2 satisfies w ; then there exists an s_3 accessible from s which satisfies w . By transitivity s_2 is also accessible from s and we may take $s_3 = s_2$.

Having a list of valid formulas, it is natural to look for an axiomatic system in which we take some of these formulas as basic axioms and provide a set of sound inference rules by which we hope to be able to prove other valid formulas as theorems. In order to denote the fact that a formula w is a theorem derivable in our logical system we will write $\vdash w$. This will be the case if w is an axiom or derivable from the axioms by a proof using the inference rules of the system.

Axioms:

A1. $\vdash \Diamond \sim w \equiv \sim \Box w$

A2. $\vdash \Box(w_1 \supset w_2) \supset (\Box w_1 \supset \Box w_2)$

A3. $\vdash \Box w \supset w$

A4. $\vdash \Box w \supset \Box\Box w$.

The inference rules are:

R1. If w is an instance of a propositional tautology, then $\vdash w$ (Tautology Rule)

R2. If $\vdash w_1 \supset w_2$ and $\vdash w_1$ then $\vdash w_2$ (Modus Ponens)

R3. If $\vdash w$ then $\vdash \Box w$. (Modal Generalization)

All these rules are sound. The soundness of R1 and R2 is obvious. Note that in R1 we also include modal instances of tautologies, e.g., $\Box w \supset \Box w$. To justify R3 we recall that validity of w means that w is true in all states of every universe, hence $\Box w$ is also valid.

This system provides a logical basis for propositional reasoning. In the Modal Logic circles this system is known as S4 (see, e.g., [H&C]).

Some theorems which can be derived in that system are:

T1. $\vdash w \supset \Diamond w$

T2. $\vdash \Box(w_1 \wedge w_2) \equiv \Box w_1 \wedge \Box w_2$

T3. $\vdash \Box(w_1 \supset w_2) \supset (\Diamond w_1 \supset \Diamond w_2)$

T4. $\vdash \Diamond(w_1 \vee w_2) \equiv \Diamond w_1 \vee \Diamond w_2$.

Note that because of the universal character of $\Box$ it commutes with $\wedge$, while $\Diamond$ which is existential commutes with $\vee$.

T5. $\vdash \Diamond(w_1 \wedge w_2) \supset \Diamond w_1 \wedge \Diamond w_2$

T6. $\vdash (\Box w_1 \vee \Box w_2) \supset \Box(w_1 \vee w_2)$

T7. $\vdash \Box w_1 \wedge \Diamond w_2 \supset \Diamond(w_1 \wedge w_2)$

T8. $\vdash \Box w \equiv \Box\Box w$

T9. $\vdash \Diamond w \equiv \Diamond\Diamond w$.

Because of these last two theorems we can collapse any string of consecutive identical modalities such as $\Box\ldots\Box$ or $\Diamond\ldots\Diamond$ into a single modality of the same type.

Since we intend to use predicates in our reasoning we have to extend our system to include some axioms and rules involving quantifiers and their interaction with modalities:

P1. $\vdash (\forall x w(x)) \supset w(t)$

where t is any term "free for x" in w .

P2. $\vdash (\forall x \Box w) \supset (\Box \forall x w)$ (Barcan's Formula).

The last implies the commutativity of $\forall$ with $\Box$, both having universal character with one quantifying over individuals while the other quantifying over states.

An additional rule of inference is:

R4. If $\vdash w_1 \supset w_2$ then $\vdash w_1 \supset \forall x w_2$

provided w_1 does not contain free occurrences of x .

Some theorems of the predicate modal system are:

T10. $\vdash (\forall x \Box w) \equiv (\Box \forall x w)$

T11. $\vdash (\exists x \Diamond w) \equiv (\Diamond \exists x w)$.

The system consisting of axioms A1 - A4, P1, P2, and rules R1 - R4 has been shown to be complete (see [H&C]).

In this next section we consider the application of the general modal framework to the analysis of programs. For the class of universes which are used there, the states in a given universe all share the same domain D and may differ by at most the values assigned to proposition symbols and individual variables. Such restricted universes are called D-universes. Since in such universes the assignment to all the other symbols is common to all states, we may associate this common part of the interpretation with the universe itself rather than with each state. Thus, a D-universe can be defined to consist of: The domain D , a common partial D-interpretation, a set of states each of which gives an assignment to the rest of the proposition symbols and individual variables, and an accessibility relation on the states. Typical D domains are the domain N of natural numbers, the domain Z of integers, the domain R of real numbers, the domain L of lists, the domain T of trees, etc.

A formula w over a domain D is any partially interpreted modal formula which may contain concrete predicates, functions, and individual elements over D , as well as uninterpreted predicate symbols, function symbols, individual constants and individual variables. A formula which is true in all states of all D-universes for a fixed D is called a D-valid formula.

The following are some examples of D-valid formulas for different D's : Each instance of the formula schema:

$$A(0) \wedge \forall n[A(n) \supset A(n+1)] \supset A(k)$$

is an N-valid formula. This partially interpreted formula schema represents the induction principle over the natural numbers.

Similarly, each instance of the schema:

$$\forall t\,[(\forall t' \prec t)\ A(t') \supset A(t)] \supset A(t)$$

is a T-valid formula, where '$\prec$' denotes the subtree relation between trees. This states the complete induction principle over trees.

II. MODAL LOGIC APPLIED TO PROGRAM ENVIRONMENT

In this section we apply the general concepts of Modal Logic to situations generated by the execution of programs. To simplify the presentation we will only consider deterministic programs. The power and elegance of the modal method are even more pronounced in dealing with nondeterministic and parallel programs.

For the concept of a state we will take an "execution state" which consists of the current values of all program variables at a certain stage in the execution. The accessibility relation between execution states will represent derivability by the program's execution. We will use predicates nd quantifiers to describe properties of a single state and modalities to desc[illegible] properties of the execution leading from one state to another.

Let [illegible] consider some particular program A with n program variables $\bar{y} = y_1,\ldots,y_n$. Assume that the program operates over a domain D . Let $\ell_0,\ell_1,\ldots,\ell_e$ be a set of labels, labeling every statement of the program. ℓ_0 is the single entry point and ℓ_e the single exit point. An <u>execution state</u> has the general structure $s = \langle \ell,\bar{\eta}\rangle$ with $\ell \in \{\ell_0,\ldots,\ell_e\}$ and $\bar{\eta} \in D^n$. For every input $\bar{\xi}$, the program generates an <u>execution sequence</u>:

$$\sigma = s^0, s^1, \ldots$$

where $s^0 = \langle \ell^0,\bar{\xi}\rangle$, and each s^i is an execution state.

The basic accessibility relation R holds between two states $<\ell,\bar{\eta}>$ and $<\ell',\bar{\eta}'>$ if there exists a computation path from ℓ to ℓ' which transforms $\bar{\eta}$ at ℓ to $\bar{\eta}'$ at ℓ' .

With these conventions we will proceed to express meaningful properties of programs and their executions. Remember that under our rules of the game we are never to mention R explicitly.

The formulas we will consider will use a basic vocabulary which includes a set of special propositions:

$$at\ell_0,\ at\ell_1,\ldots,\ at\ell_e,$$

each corresponding to one of the labels. In addition we will allow arbitrary predicates over the $\bar{y}$ (program variables) and additional auxiliary variables $\bar{u}$. We assume that only the $\bar{y}$'s change from one state to another, while the $\bar{u}$'s , being external, remain fixed. Let $\bar{\zeta}$ denote the fixed values of the auxiliary variables. The truth value of an atomic formula at a state $s = <\ell,\bar{\eta}>$ is given as follows:

$$at\ell_i \text{ is true at } s \text{ iff } \ell_i = \ell \ .$$
$$p(\bar{y},\bar{u}) \text{ is true at } s \text{ iff } p(\bar{\eta},\bar{\zeta}) = \underline{true} \ .$$

The truth value of a non-atomic formula, possibly containing modalities, is determined by the classic rules and the rules for interpreting modalities given above.

Note that our definition of a state here conforms with the general convention of D-universes. The specification of a state only specifies the elements by which one state may differ from another, namely, propositions $(at\ell_0,at\ell_1,\ldots,at\ell_e)$ and the values assigned to some of the individual free variables $(y_1,\ldots,y_n)$.

1. Invariance Properties

Consider first the class of program properties which are expressible by formulas of the form

$$w_0 \supset \Box w \ .$$

In the general modal context such a formula claims that w holds true in all states R-accessible from any state satisfying w_0 . In our programming context we will often take w_0 as $at\ell_0 \wedge \bar{y} = \bar{\xi}$, which exactly characterizes the initial state, and then we have

$$(at\ell_0 \wedge \bar{y} = \bar{\xi}) \supset \Box w \ .$$

Then this states that w is true for all states arising during execution. A formula of this form therefore expresses an invariance property.

Samples of important properties which fall under this category are:

A. Partial Correctness. Let $\varphi(\bar{x})$ be a precondition which restricts the set of inputs for which the program is supposed to be correct, and $\psi(\bar{x},\bar{y})$ the statement of its correctness, i.e., the relation which should hold between the input values $\bar{x}$ and the output values $\bar{y}$. Then in order to state partial correctness w.r.t. (φ,ψ) we can write:

$$(at\ell_0 \wedge \bar{y} = \bar{x} \wedge \varphi(\bar{x})) \supset \Box(at\ell_e \supset \psi(\bar{x},\bar{y})) \ .$$

This claims that if the initial state satisfies the restricting precondition, then in any state accessible from the initial state: If that state happens to be the exit state ℓ_e then $\psi(\bar{x},\bar{y})$ holds between the input values $\bar{x}$ and the current $\bar{y}$ values. Thus this formula states that all convergent φ-sequences terminate in a state satisfying ψ , but it does not guarantee termination itself.

Let us consider a concrete example (a program computing $x!$ over the natural numbers):

Program P1:

```
        $\ell_0$: $y_2 \leftarrow 1$ ;
        $\ell_1$: if  $y_1 > 0$  then
                 begin  $\ell_2$:  $(y_1,y_2) \leftarrow (y_1-1,\ y_1 \cdot y_2)$ ;
                        $\ell_3$:  goto  $\ell_1$
                 end ;
        $\ell_e$: Halt.
```

The statement of its partial correctness is

$$(at\ell_0 \wedge y_1 = x \wedge x \geqslant 0) \supset \square(at\ell_e \supset y_2 = x!) \ .$$

This is indeed an inherently invariant property since it is actually only a part of a bigger global invariant which represents the "network of invariants" normally used in the Invariant-Assertion Method (see [FLO]), namely:

$$\begin{aligned}\square(at\ell_0 \wedge y_1=x \wedge x\geqslant 0) \supset [&(at\ell_1 \supset y_1\geqslant 0 \wedge y_2 \cdot y_1!=x!) \wedge \\ &(at\ell_2 \supset y_1>0 \wedge y_2 \cdot y_1!=x!) \wedge \\ &(at\ell_3 \supset y_1\geqslant 0 \wedge y_2 \cdot y_1!=x!) \wedge \\ &(at\ell_e \supset y_1=0 \wedge y_2=x!)] \ .\end{aligned}$$

B. Clean Behavior. For every location in a program we can formulate a cleanness condition which states that the statement at this location will execute successfully and generate no fault. Thus if the statement contains division, the cleanness condition will include the clause that the divisor is nonzero or not too small to avoid arithmetic overflow. If the statement contains an array reference, the cleanness condition will imply that the subscript expressions do not exceed the declared range. Denoting the cleanness condition at location i by α_i, the statement of clean behavior is:

$$(at\ell_0 \wedge \varphi(\bar{y})) \supset \Box(\bigwedge_i (at\ell_i \supset \alpha_i)) .$$

The conjunction is taken over all "potentially dangerous" locations in the program.

For example, the program P1 above should produce only natural number values during its computation. A cleanness condition at ℓ_2, which is clearly a critical point, is:

$$(at\ell_0 \wedge y_1 \geq 0) \supset \Box(at\ell_2 \supset y_1 > 0)$$

guaranteeing that the subtraction at ℓ_2 always yields a natural number.

C. <u>Global Invariants</u>. Very frequently, cleanness conditions are not related to any particular location. More generally, some other properties may be "truly" invariant independent of the location. In these cases we speak of <u>global invariants</u> unattached to any particular location. The expression of global invariance is even more straightforward. Thus to claim for the example above that y_1 is always a natural number, we may write:

$$(at\ell_0 \wedge y_1 \geq 0 \wedge \text{integer}(y_1)) \supset \Box(y_1 \geq 0 \wedge \text{integer}(y_1)) .$$

Another global invariant valid for this example is:

$$(at\ell_0 \wedge (y_1, y_2) = (x, 1)) \supset \Box(y_2 \cdot y_1! = x!) ,$$

which states that <u>everywhere</u> in the execution $y_2 \cdot y_1! = x!$.

Similarly, to ensure subscript cleanness we may claim global invariants of the form:

$$(at\ell_0 \wedge \varphi(\bar{y})) \supset \Box(0 \leq I \leq N) .$$

Another example of the usage of invariants is in the context of a program whose output is not necessarily apparent at the end of the execution; for example, a program whose output is printed on an external file during the computation. Consider a program for printing a sequence of prime numbers. Let ℓ be any location which contains a "print" instruction of form:

$$\ell: \text{ print}(y) .$$

Then a part of the correctness statement for such a program is:

$$w_0 \supset \Box(\text{at}\ell \supset \text{prime}(y))$$

for all print locations ℓ . It indicates that nothing but primes is printed.

Note that this property may specify the partial correctness even of <u>continuous</u> programs, i.e., programs which are not supposed to terminate but to operate continuously.

Even though our main interest in this paper is in deterministic programs, we cannot resist illustrating the efficacy of the modal formalism for parallel programs.

A state in the execution of two parallel processes will be structured as: $s = \langle \ell_1, \ell_2; \bar{\eta} \rangle$, i.e., it will contain references to locations in <u>both</u> processes. These references are tested by the propositions $\text{at}\ell_1$, $\text{at}\ell_2$ for all locations ℓ_1 and ℓ_2 in the two processes.

D. <u>Mutual Exclusion</u>. Let us consider first the property of Mutual Exclusion. Let two processes P_1 and P_2 execute in parallel. Assume that each process contains a section C_i , $i = 1,2$, which includes some task critical to the cooperation of the two processes. For example, it might access a shared device (such as a disk) or a shared variable. If the nature of the task is such that it must be done exclusively by one process or the other, but never by both of them simultaneously, we call these sections critical sections. The property that states that the processes are never simultaneously executing in their respective critical sections is called Mutual Exclusion with respect to this pair of critical sections.

The property of mutual exclusion for C_1 and C_2 can be described by:

$$w_0 \supset \square(\sim(\mathrm{at}\ell_1 \wedge \mathrm{at}\ell_2))$$

for every pair of labels $\ell_1 \in C_1$ and $\ell_2 \in C_2$. This states that it is never the case that the joint execution of the processes reaches ℓ_1 and ℓ_2 simultaneously. Hence, mutual exclusion is implied. In practice, one does not have to actually consider all possible pairs $\ell_i \in C_i$.

E. Deadlock Freedom. A standard synchronization device in concurrent systems is the semaphore which is implemented by the atomic instructions:

$$p(x):\quad x>0 \rightarrow [x \leftarrow x-1]$$
$$v(x):\quad x \leftarrow x+1 .$$

A process reaching a $p(x)$ instruction will proceed beyond it only if $x>0$ and then it will decrement x by 1, usually setting it to 0. No further process may go beyond a $p(x)$ instruction until somebody (in all probability the process that has just decremented x) will perform a $v(x)$ operation, increasing x to 1.

A concurrent system consisting of n parallel processes is said to be deadlocked if none of the processes can execute any further step. If we assume that the only synchronization device in a system is semaphores, then the only possibility for a deadlock is the situation:

$$\begin{array}{ccc} \vdots & \qquad\qquad & \vdots \\ \ell_1 : p(x^1) & & \ell_n : p(x^n) \\ \vdots & & \vdots \end{array}$$

for some locations $\ell_1, \ldots, \ell_n$ (ℓ_i belonging to process i), where all n of the processes in the system are currently waiting for 'p' operations on the semaphore variables $x^1, \ldots, x^n$ (not necessarily distinct) while $x^1 = x^2 = \ldots = x^n = 0$.

To exclude this possibility we can require:

$$w_0 \supset \Box\left(\bigwedge_{i=1}^{n} at\ell_i \supset \bigvee_{i=1}^{n} (x^i > 0)\right) .$$

This requires that whenever all the processes are each at the ℓ_i: $p(x^i)$ operation, $i = 1,\ldots,n$, at least one of the x^i's must be positive. The corresponding process can then proceed.

In order to completely eliminate the possibility of deadlock in the system, we must impose a similar requirement for every n-tuple of 'p' locations.

2. <u>Eventuality Properties</u>

A second category of properties are those expressible by formulas of the form:

$$w_1 \supset \Diamond w_2 .$$

In the general context this means that if at any state s_1 , w_1 is true, there exists a state s_2 , R-accessible from s_1 , in which w_2 is true. In the programming context it means that if w_1 ever arises during execution, it will eventually be followed by another state in which w_2 is true. A formula of this form therefore expresses an eventuality property. Following are some samples of properties expressible by formulas of this form.

A. Total Correctness. A program is said to be totally correct w.r.t. a specification (φ,ψ) , if for every input $\bar{\xi}$ satisfying $\varphi(\bar{\xi})$, termination is guaranteed, and the final values $\bar{y}=\bar{\eta}$ upon termination satisfy $\psi(\bar{\xi},\bar{\eta})$. Once more, let ℓ_0 denote the entry location and ℓ_e the exit location of the program. Total correctness w.r.t. (φ,ψ) is expressible by:

$$(at\ell_0 \wedge \bar{y}=\bar{x} \wedge \varphi(\bar{x})) \supset \Diamond(at\ell_e \wedge \psi(\bar{x},\bar{y})) .$$

This says that if we have an execution sequence which begins in a state which is at location ℓ_0 and has values $\bar{y}=\bar{x}$ satisfying φ , then later in that execution sequence we are guaranteed to have a state which is at ℓ_e and satisfies $\psi(\bar{x},\bar{y})$.

For example, the statement of total correctness of the program P1 for the computation of x! is:

$$(at\ell_0 \wedge y_1=x \wedge x\geq 0) \supset \Diamond(at\ell_e \wedge y_2=x!) .$$

B. General Eventualities. Eventuality formulas enable us to express a causality relation between any two events, not only between program initialization and termination but also between events arising during the execution. This becomes especially important when discussing continuously executing programs, i.e., where termination is not expected. The general form of such an eventuality is:

$$(at\ell_1 \wedge \varphi_1) \supset \Diamond(at\ell_2 \wedge \varphi_2)$$

and it claims that whenever φ_1 arises at ℓ_1 we are guaranteed of eventually reaching ℓ_2 with φ_2 true. This is the exact formalization of the basic Intermittent-Assertion statement (see [M&W]):

"If sometimes φ_1 at ℓ_1 , then sometimes φ_2 at ℓ_2 ."

Consider for example the program for printing successive prime numbers. Under the invariance properties we expressed the claim that nothing but primes are printed. Here we can state that the proper sequence of primes is produced. Let

$$\ell\text{: print(y)}$$

be the only printing instruction in the program. Then the following two clauses ensure the desired property:

$$at\ell_0 \supset \Diamond(at\ell \wedge y=2)$$
$$(at\ell \wedge y=x) \supset \Diamond(at\ell \wedge y=nextprime(x)) .$$

The first statement assures arrival at ℓ with y being the first prime. The second claim ensures that after any prime is printed the next prime in sequence will eventually be printed.

Note that these statements do not guarantee that some primes are not printed more than once or out of sequence, but they do guarantee that all printed results are primes, and that a subsequence of the printed results is the ascending sequence of primes.

Again, let us allow ourselves a short excursion into the world of parallel programs.

C. Accessibility. Consider again a process which has a critical section C . In the previous discussion we have shown how to state exclusion or protection for that section. A related property is that of accessibility, that if a process wishes to enter its critical section, it will eventually get there and will not be indefinitely held up by the protection mechanism.

Let ℓ_1 be a location just before the critical section. The fact that the process is at ℓ_1 indicates an intention to enter the critical section. Let ℓ_2 be a location inside the critical section. The property of accessibility can then be expressed by:

$$at\ell_1 \supset \Diamond at\ell_2 \ ;$$

namely, whenever the program is at ℓ_1, it will eventually get to ℓ_2.

A correct construction of critical sections should ensure these two complementary properties: that of protection (exclusiveness) and that of accessibility.

D. Responsiveness. Consider an example of a program modeling an operating system. Assume that it serves a number of customer programs by scheduling a shared resource between them. Let the customer programs communicate with the operating system concerning a given resource via a set of boolean variables $\{r_i, g_i\}$. r_i is set to true by customer program number i to signal a request for the resource. g_i is set to true by the operating system to signal that customer i is granted the use of the resource. The statement that the operating system fairly responds to user requests -- responsiveness -- is given by:

$$r_i \supset \Diamond g_i$$

i.e., whenever r_i becomes true, eventually g_i will turn true.

Note that since these events are global and not attached to any specific location, they can model external events such as interrupts and unsolicited signals.

III. PROOF SYSTEMS

After giving some evidence of the power of the modal notation in expressing interesting program properties, we should search next for proof systems in which these properties can be formally established. Obviously, the basis for all such systems will be the general S4 framework introduced above. However, this basis must be augmented by additional axioms and rules, reflecting the properties of the domain and the structure of the program under consideration. These additional conditions will constrain the accessibility relation $R(s,s')$ to represent the relation of s' being derivable from s by an execution of the program. This releases us from the need to express program text syntactically in the system; instead all necessary information is captured by the constraints on the accessibility relation as expressed by the additional axioms.

Our proof systems will therefore consist of three parts: a general part which contains S4-like axioms, elaborating the general properties of the relation R; a proper part which gives an axiomatic description of the domain; and a local part consisting of axiom schemata which generate a set of local axioms for any particular program. The local axioms constrain the state sequences to those considered to be execution sequences of the program under study.

1. The sometime system.

Our simpler system, called here the sometime system, is based on S4.

A. General Part: The general part consists of the following S4 axioms:

A1. $\vdash \Diamond \sim w \equiv \sim \Box w$

A2. $\vdash \Box(w_1 \supset w_2) \supset (\Box w_1 \supset \Box w_2)$

A3. $\vdash \Box w \supset w$

A4. $\vdash \Box w \supset \Box\Box w$

P1. $\vdash (\forall x\ w(x)) \supset w(t)$

where t is "free for x" in w .

P2. $\vdash (\forall x \Box w) \supset (\Box \forall x w)$.

The rules of the inference are:

R1. If w is an instance (possibly modal) of a tautology, then $\vdash w$.

R2. If $\vdash w_1 \supset w_2$ and $\vdash w_1$, then $\vdash w_2$.

R3. If $\vdash w$, then $\vdash \Box w$.

R4. If $\vdash w_1 \supset w_2$, then $\vdash w_1 \supset \forall x w_2$

provided w_1 does not contain free occurrences of x .

This system generally constrains R to be reflexive (A3) and transitive (A4).

B. Proper part: The next part of the system contains a set of proper axioms and axiom schemata. These axioms specify all the needed properties of the domain of interest. Thus, to reason about programs manipulating natural numbers, we need the set of Peano axioms. To reason about trees we need a set of axioms giving the basic properties of trees and of the basic operations defined on them. An essential axiom schema for every domain should be the induction axiom schema. This (and all other schemata) should be formulated to admit modal instances as subformulas. Thus the induction principle for natural numbers is:

$$\vdash A(0) \wedge \forall n[A(n) \supset A(n+1)] \supset A(k) \quad .$$

A modal instance of this principle which will be used later is:

Induction Theorem:

$$\vdash \Box(P(0) \supset \Diamond\psi) \wedge \forall n[\Box(P(n) \supset \Diamond\psi) \supset \Box(P(n+1) \supset \Diamond\psi)] \supset \Box(P(k) \supset \Diamond\psi) .$$

Similar induction theorems will exist for any other set of proper axioms which depend on natural well-founded orderings existing in the domain.

C. Local part: The axioms and rules above represent the general framework needed for our reasoning. Next we introduce a set of local axioms which depend on the particular program to be analyzed.

The first axiom depends only on the identity of the program variables. Let w be any formula which does not contain any program variables or propositions $at\ell_i$, then the following is an axiom:

Frame Axiom: $\vdash w \supset \Box w$.

The justification hinges on the fact that R-related states may differ from one another only in the assignments to program variables.

A second generic axiom states that every state s has exactly one label ℓ_i such that $at\ell_i$ is true.

Location Axiom: $\vdash \sum_{i=0}^{e} at\ell_i = 1$.

We use here the abbreviation $\Sigma p_i = 1$ or $p_1 + \ldots + p_n = 1$ meaning that exactly one of the p_i's is true.

The other axioms are local to each program. For these axiom schemata we make the following simplifying assumptions about the program:

Assume that the program is represented as a directed graph whose nodes are the program locations or labels, and whose edges represent transitions between the labels. A transition is an instruction of the general form

$$c(\bar{y}) \rightarrow [\bar{y} \leftarrow \bar{f}(\bar{y})] \ .$$

$c(\bar{y})$ is a condition (may be the trivial condition <u>true</u>) under which the transition replacing $\bar{y}$ by $\bar{f}(\bar{y})$ should be taken. $\bar{y} = y_1, \ldots, y_n$ is the vector of program variables. We assume that all the conditions $c_1, \ldots, c_k$ on transitions departing from any node are mutually exclusive and exhaustive (i.e., $\Sigma\, c_i = 1$).

The role of the local axioms is to introduce our knowledge about the program into the system. Since the system does not provide direct tools for speaking about programs (such as Hoare's formalism), the local axioms represent the program by characterizing the possible state transitions under the program control.

For any transition:

$$\begin{tikzcd} \textcircled{\ell} \arrow[r, "c(\bar{y}) \rightarrow [\bar{y} \leftarrow f(\bar{y})]", "\alpha"'] & \textcircled{\ell'} \end{tikzcd}$$

we can generate an axiom F_α . This axiom corresponds to a "forward" propagation (derivation of the strongest postcondition) across the transition α :

$$F_\alpha : \vdash [at\ell \wedge c(\bar{y}) \wedge \bar{y}=\bar{\eta}] \supset \Diamond(at\ell' \wedge \bar{y}=\bar{f}(\bar{\eta})) \ .$$

This axiom states: If at any state, execution is at ℓ , $c(\bar{y})$ hold, and the current values of $\bar{y}$ are $\bar{\eta}$, then sometime later we will be at ℓ' with the variables $\bar{y} = \bar{f}(\bar{\eta})$.

A different approach which suggests an alternate axiom schema is obtained by "backward" substitution (derivation of the weakest precondition):

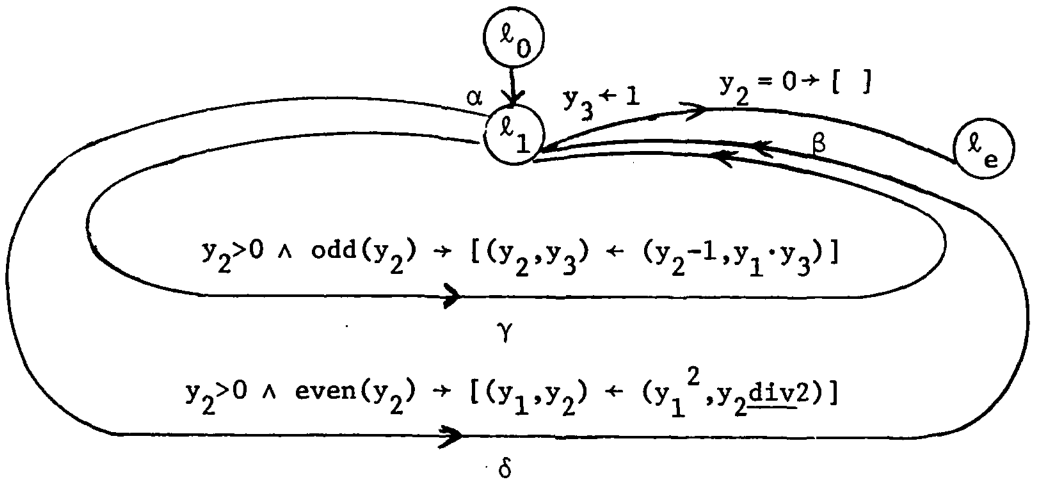
$$B_\alpha : \vdash [at\ell \wedge c(\bar{y}) \wedge P(\bar{f}(\bar{y}))] \supset \Diamond(at\ell' \wedge P(\bar{y})) \ ,$$

where $P(\bar{f}(\bar{y}))$ denotes the substitution of $\bar{f}(\bar{y})$ for all free occurrences of $\bar{y}$ in $P(\bar{y})$. This form of the axiom expresses the effect of the transition on an arbitrary predicate P (predicate transformer). It says that if $at\ell \wedge c(\bar{y})$ and $P(\bar{f}(\bar{y}))$ hold, then we are guaranteed to eventually reach ℓ' with $P(\bar{y})$. F_α and B_α are equivalent and can be derived from each other.

Note that both forms ignore the fact that the 'sometime' guaranteed is actually in the immediately next instance. This is a consequence of the fact that we can only guarantee things eventually and have no way to formulate properties of the next instance.

Consider for example the following program over the integers which raises a number x_1 to an integral power $x_2 \geq 0$, assuming that (x_1,x_2) are the initial values of the variables (y_1,y_2).

<u>Program P2</u>:



ℓ_0
α
$y_3 \leftarrow 1$
ℓ_1
$y_2 = 0 \rightarrow [\ ]$
β
ℓ_e
$y_2>0 \wedge odd(y_2) \rightarrow [(y_2,y_3) \leftarrow (y_2-1,y_1 \cdot y_3)]$
γ
$y_2>0 \wedge even(y_2) \rightarrow [(y_1,y_2) \leftarrow (y_1^2,y_2 \underline{div} 2)]$
δ

The local backward axiom schemata corresponding to this program are:

B_α : $\vdash [at\ell_0 \wedge P(y_1,y_2,1)] \supset \Diamond(at\ell_1 \wedge P(y_1,y_2,y_3))$

B_β : $\vdash [at\ell_1 \wedge y_2=0 \wedge P] \supset \Diamond(at\ell_3 \wedge P)$

B_γ : $\vdash [at\ell_1 \wedge y_2>0 \wedge odd(y_2) \wedge P(y_1,y_2-1,y_1 \cdot y_3)] \supset \Diamond(at\ell_1 \wedge P(y_1,y_2,y_3))$

B_δ : $\vdash [at\ell_1 \wedge y_2>0 \wedge even(y_2) \wedge P(y_1^2,y_2\underline{div}2,y_3)] \supset \Diamond(at\ell_1 \wedge P(y_1,y_2,y_3))$

D. <u>Derived rules</u>: Before demonstrating a proof in the system we will develop several useful derived rules:

<u>$\Box\Box$-Generalization</u>: $\dfrac{\vdash P \supset Q}{\vdash \Box P \supset \Box Q}$.

This is obtained by application of modal generalization R3 and the use of A2.

By substituting in the above $\sim Q$ for P and $\sim P$ for Q, we obtain:

<u>$\Diamond\Diamond$-Generalization</u>: $\dfrac{\vdash P \supset Q}{\vdash \Diamond P \supset \Diamond Q}$.

The following additional rules correspond to proof rules existent in most axiomatic verification systems. (In these rules interpret $P \supset \Box Q$ and $P \supset \Diamond Q$ as stating the partial and total correctness of some program segment respectively.)

<u>Consequence</u>: $\dfrac{\vdash P \supset Q,\ \vdash Q \supset \Diamond R,\ \vdash R \supset S}{\vdash P \supset \Diamond S}$.

From $\vdash R \supset S$ (using $\Diamond\Diamond$-Gen.) we obtain $\vdash \Diamond R \supset \Diamond S$ which can be combined with the other premises to lead to the result.

<u>Concatenation</u>: $\dfrac{\vdash P \supset \Diamond Q,\ \vdash Q \supset \Diamond R}{\vdash P \supset \Diamond R}$.

Here we derive $\vdash \Diamond Q \supset \Diamond\Diamond R$ by the $\Diamond\Diamond$-Gen rule. We then use Theorem T9 ($\vdash \Diamond\Diamond R \supset \Diamond R$) to obtain $\vdash \Diamond Q \supset \Diamond R$. The conclusion follows by propositional reasoning.

A derived frame rule more appropriate to step-by-step transitions is given by:

<u>Frame Rule</u>: $\dfrac{\vdash P \supset \Diamond Q}{\vdash (P \wedge w) \supset \Diamond(Q \wedge w)}$

provided w contains no program variables or propositions $at\,\ell_1$.

This rule is a simple consequence of the Frame Axiom $\vdash w \supset \Box w$ and of Theorem T7 ($\vdash \Diamond Q \wedge \Box w \supset \Diamond(Q \wedge w)$) .

We will also need some rules for establishing the convergence of loops. These rules will of course depend on the domain under discussion and the induction principle provided in that domain. For the domain of natural numbers we already mentioned the Induction Theorem:

$$\Box[P(0) \supset \Diamond\psi] \wedge \forall n\, [\Box(P(n) \supset \Diamond\psi) \supset \Box(P(n+1) \supset \Diamond\psi)] \supset \Box[P(k) \supset \Diamond\psi] .$$

Using this induction theorem we can derive the following rule:

<u>Induction Rule 1</u>: $\dfrac{\vdash P(0) \supset \Diamond\psi, \quad \vdash \Box(P(n) \supset \Diamond\psi) \supset \Box(P(n+1) \supset \Diamond\psi)}{\vdash (\exists k P(k)) \supset \Diamond\psi}$.

This rule says that if $P(0)$ eventually guarantees ψ , and if for any n , the fact that $P(n)$ guarantees ψ implies that $P(n+1)$ guarantees ψ , then if $P(k)$ is true for some k , ψ is eventually guaranteed. This rule is useful for proving convergence of a loop, if for example we have a P such that $P(0) \supset \Diamond\psi$ and across the loop's body $P(n+1) \supset \Diamond P(n)$, implying the second premise of the rule.

From this rule we can derive a more liberal form of the induction rule.

<u>Induction Rule 2</u>: $\dfrac{\vdash P(0) \supset \Diamond\psi, \quad \vdash P(n+1) \supset \Diamond(\psi \vee P(n))}{\vdash (\exists k P(k)) \supset \Diamond\psi}$.

Rule 2 is more liberal than Rule 1 since it does not require us to give an exact estimate of the number of repetitions of the loop, but allows instead an estimate of an upper bound. We can see this by observing that in the previous case we required that $P(n+1)$ leads to $P(n)$ across the loop's body, and only $P(0)$ ensures ψ . Thus to start the argument we have to state $P(k)$ where we expect the loop to be executed k times. In Rule 2 we claim that for each n, either $P(n+1)$ implies $P(n)$ across the loop, or that it establishes ψ and no further execution is necessary. Thus $P(k)$ ensures that either the loop is executed at most k times and ψ is established on the last iteration or earlier.

2. Total Correctness - Example and Discussion

Let us use this system to establish the correctness of the example program P2 computing $x_1^{x_2}$. We will prove that

$$\vdash [at\ell_0 \wedge (y_1,y_2)=(x_1,x_2) \wedge x_2 \geq 0] \supset \Diamond(at\ell_3 \wedge y_3 = x_1^{x_2}) ,$$

namely: If we are in any state at ℓ_0 with $\bar{y}=\bar{x}$ then there exists a state in which we are at ℓ_e and $y_3 = x_1^{x_2}$.

In the proof below we use the backward form of the axioms. The proof proceeds as follows:

1. $\vdash [at\ell_0 \wedge (y_1,y_2)=(x_1,x_2) \wedge x_2 \geq 0] \supset [at\ell_0 \wedge y_2 \geq 0 \wedge y_1^{y_2} = x_1^{x_2}]$

   A Z-valid formula.

2. $\vdash [at\ell_0 \wedge y_2 \geq 0 \wedge 1 \cdot y_1^{y_2} = x_1^{x_2}] \supset \Diamond[at\ell_1 \wedge y_2 \geq 0 \wedge y_3 \cdot y_1^{y_2} = x_1^{x_2}]$

   By B_α with $P(y_1,y_2,y_3) = (y_2 \geq 0 \wedge y_3 \cdot y_1^{y_2} = x_1^{x_2})$.

3. $\vdash [at\ell_0 \wedge (y_1,y_2)=(x_1,x_2) \wedge x_2 \geq 0] \supset \Diamond[at\ell_1 \wedge y_2 \geq 0 \wedge y_3 \cdot y_1^{y_2} = x_1^{x_2}]$

   By Consequence 1,2.

Denote now:

$$Q(n,\bar{y}):\ at\ell_1 \wedge 0 \leq y_2 \leq n \wedge y_3 \cdot y_1^{y_2} = x_1^{x_2} .$$

Using Induction Rule 2, we will establish

(*) $\vdash (\exists k Q(k,\bar{y})) \supset \Diamond(at\ell_e \wedge y_3 = x_1^{x_2})$,

where we take $\psi = (at\ell_e \wedge y_3 = x_1^{x_2})$.

Applying the Consequence Rule to 3, we have:

4. $\vdash [at\ell_0 \wedge (y_1,y_2) = (x_1,x_2) \wedge x_2 \geq 0] \supset \Diamond Q(y_2,\bar{y})$

which establishes $\exists k Q(k,\bar{y})$ by taking $k = y_2$.

In order to use the Induction Rule 2, we show first $Q(0,\bar{y}) \supset \Diamond\psi$: Note that $Q(0,\bar{y})$ implies $y_2 = 0$.

5. $\vdash Q(0,\bar{y}) \supset \Diamond[at\ell_3 \wedge y_3 = x_1^{x_2}]$, hence $\Diamond\psi$, by B_β and Consequences.

We now proceed to show by case analysis that

$$\vdash Q(n+1,\bar{y}) \supset \Diamond[\psi \vee Q(n,\bar{y})] .$$

6. $\vdash [Q(n+1,\bar{y}) \wedge y_2 = 0] \supset \Diamond[at\ell_3 \wedge y_3 = x_1^{x_2}]$, hence $\Diamond\psi$,

by B_β and Consequences.

7. $\vdash [Q(n+1,\bar{y}) \wedge y_2 > 0 \wedge odd(y_2)] \supset \Diamond Q(n,\bar{y})$

by B_γ , logic and Consequences.

8. $\vdash [Q(n+1,\bar{y}) \wedge y_2 > 0 \wedge even(y_2)] \supset \Diamond Q(n,\bar{y})$

by B_δ , logic and Consequences.

In the proof of 8 we use the fact that $0 < y_2 \leq n+1$ implies $0 \leq y_2 \underline{div} 2 \leq n$.

9. $\vdash Q(n+1,\bar{y}) \supset \Diamond[(at\ell_e \wedge y_3 = x_1^{x_2}) \vee Q(n,\bar{y})]$

by taking the "or" of 6, 7, 8, propositional reasoning and T4.

By the Induction Rule 2 we get from 5 and 9:

10. $\vdash \exists kQ(k,\bar{y}) \supset \Diamond[at\ell_3 \wedge y_3 = x_1^{x_2}]$.

Combining 4 and 10 with Concatenation and Consequences, we get:

11. $\vdash [at\ell_0 \wedge (y_1,y_2) = (x_1,x_2) \wedge x_2 \geq 0] \supset \Diamond[at\ell_3 \wedge y_3 = x_1^{x_2}]$.

This concludes the proof of total correctness of our example program.

Clearly, a statement of the form

$$[at\ell \wedge P] \supset \Diamond[at\ell' \wedge P']$$

is exactly a formalization of the typical "intermittent assertion":

"If <u>sometime</u> P at ℓ then <u>sometime</u> P' at ℓ' ."

Thus we are justified in regarding this modal system as the most appropriate formalization of the Intermittent-Assertion method.

When we investigate the "power" of the system we find that it is adequate for proving valid eventualities, i.e., properties of the form:

$$P \supset \Diamond Q$$

which are valid for programs over the given domain. For this reason we named this system the "sometime" system.

Unfortunately this system is inadequate for proving invariance properties such as partial correctness and global properties. This deficiency is not a flaw in the logic formalism itself, bit in the failure of the local axioms to capture exactly the execution sequences of the given program and nothing more. While $[at\ell \wedge P] \supset \Diamond[at\ell' \wedge P']$ guarantees that ℓ' will be reached sometime in the future, we have no way to specify that ℓ' is actually reached in the next immediate state. This does not hurt us when we prove eventualities since we do not care about intermediate states other than those explicitly mentioned. But in order to claim invariance, we have to keep track of all intermediate states, and then we must be able to describe what happens in the next immediate state.

3. The Nexttime System

In order to correct this deficiency we introduce an additional modal operator into our system. This is the next instance operator, denoted by $\bigcirc$.

A semantic model for the extended system will now consist of a set of states and an immediate accessibility relation ρ connecting some of these states. ρ corresponds to the next or immediate future relation. In any such universe (model) we define R to be the reflexive transitive closure of ρ which therefore gives it the meaning of "present or eventual future". Semantic truth in a state s in such a universe is now defined (extending the previous definition) as:

$$|\Box w|_s \equiv \forall s'[R(s,s') \supset |w|_{s'}]$$
$$|\Diamond w|_s \equiv \exists s'[R(s,s') \wedge |w|_{s'}]$$
$$|\bigcirc w|_s \equiv \exists s'[\rho(s,s') \wedge |w|_{s'}]$$

This extended system is aptly called the nexttime system.

Following we present an axiomatic system for the 'nexttime' logic. Where it differs very little from the 'sometime' system, we will only mark the differences.

A. General Part:

Axioms:

C1. $\vdash \Diamond \sim w \equiv \sim \Box w$

C2. $\vdash \Box(w_1 \supset w_2) \supset (\Box w_1 \supset \Box w_2)$

C3. $\vdash \Box w \supset w$

C4. $\vdash O(\sim w) \equiv \sim O(w)$

C5. $\vdash O(w_1 \supset w_2) \supset (O w_1 \supset O w_2)$

C6. $\vdash \Box w \supset O w$

C7. $\vdash \Box w \supset O \Box w$

C8 $\vdash \Box (w \supset O w) \supset (w \supset \Box w)$

P1. $\vdash [\forall x w(x)] \supset w(t)$ where t is "free for x" in w .

P2 $\vdash (\forall x \Box w) \supset (\Box \forall x w)$

P3 $\vdash (\forall x O w) \supset (O \forall x w)$.

C1-C3, P1, P2 are the same as A1-A3, P1, P2 in the 'sometime' system. C4 claims the uniqueness of the next instance. C5 is the analogue of C2 for the O operator. C6 claims that the next state is one of the reachable states. It also guarantees that each state has a successor. (In order to satisfy this requirement in the programming context we stipulate that each exit label in the program's graph is connected to itself by a trivial transition.) C7 is a weaker version of A4 ($\vdash \Box w \supset \Box \Box w$) in the 'sometime' system and can be used together with C8 to prove this as a theorem in the 'nexttime' system. C8 is the "computational induction" axiom; it states that if a property is inherited over one step transition, it is invariant over any path.

Rules of Inference: Identical to R1-R4 of the 'sometime' system.

A simple theorem of this system is:

T12: $\vdash O w \supset \Diamond w$

obtained by negation of C6 and applications of C1 and C4.

B. <u>Proper part</u>: Since the proper part consists solely of first-order axioms, it is identical with the proper part of the 'sometime' system.

C. <u>Local part</u>: The Frame and Location Axioms remain the same. The main difference is in the local axioms which now describe transitions between a state and its immediate successor. For a transition

$$\begin{tikzcd} \ell \arrow[r, "c(\bar{y}) \rightarrow [\bar{y} \leftarrow \bar{f}(\bar{y})]", "\alpha"'] & \ell' \end{tikzcd}$$

we generate the "forward" axiom $\tilde{F}_\alpha$:

$$\tilde{F}_\alpha: \ \vdash \ [at\ell \wedge c(\bar{y}) \wedge \bar{y} = \bar{\eta}] \supset \bigcirc(at\ell' \wedge \bar{y} = \bar{f}(\bar{\eta})) \ ,$$

and similarly the "backward" axiom schema:

$$\tilde{B}_\alpha: \ \vdash \ [at\ell \wedge c(\bar{y}) \wedge P(\bar{f}(\bar{y}))] \supset \bigcirc(at\ell' \wedge P(\bar{y})) \ .$$

By the theorem $\vdash \bigcirc w \supset \Diamond w$, we have that $\vdash \tilde{F}_\alpha \supset F_\alpha$. Therefore any proof in the 'sometime' system is automatically carried over to the 'nexttime' system. Consequently the 'nexttime' system is also adequate for proving total correctness and other eventualities. In addition it is also adequate for proving invariance properties.

4. <u>Proof of Invariance</u>

Let us consider now a typical proof of invariance. Let Q be an inductive program property. Intuitively this means that Q is true of the initial state and is preserved under any program step. Thus we have

(a) $\vdash \varphi \supset Q$

for the input predicate φ .

Also for any transition α: $c(\bar{y}) \rightarrow [\bar{y} \leftarrow \bar{f}(\bar{y})]$, we have

(b) $\vdash c(\bar{y}) \wedge Q(\bar{y}) \supset Q(\bar{f}(\bar{y}))$.

Let ℓ be any label in the program and let its outgoing transitions be α_i leading to ℓ_i respectively. Assume each transition to be $\alpha_i\colon c_i(\bar{y}) \to [\bar{y} \leftarrow \bar{f}_i(\bar{y})]$. We have already assumed that $\bigvee_i c_i(\bar{y}) = \underline{\text{true}}$.

For any i we have

$$\vdash [at\ell \wedge c_i(\bar{y}) \wedge Q(\bar{y})] \supset [at\ell \wedge c_i(\bar{y}) \wedge Q(\bar{f}_i(\bar{y}))]$$

by the inductiveness of Q , i.e., (b).

$$\vdash [at\ell \wedge c_i(\bar{y}) \wedge Q(\bar{f}_i(\bar{y}))] \supset \bigcirc(at\ell_i \wedge Q(\bar{y}))$$

by the local backward axiom $\tilde{B}_{\alpha_i}$.

Combining the last two we get:

$$\vdash [at\ell \wedge c_i(\bar{y}) \wedge Q(\bar{y})] \supset \bigcirc(at\ell_i \wedge Q(\bar{y})) ,$$

from which, by Consequence, we get:

$$\vdash [at\ell \wedge c_i(\bar{y}) \wedge Q(\bar{y})] \supset \bigcirc Q(\bar{y}) .$$

Since the above was obtained for an arbitrary i we can take the logical 'or' of all these statements over all i's . Using the fact that $\bigvee_i c_i = \underline{\text{true}}$, we obtain:

$$\vdash [at\ell \wedge Q(\bar{y})] \supset \bigcirc Q(\bar{y}) .$$

Taking the disjunction over all program labels $\ell \in \{\ell_0,\ldots,\ell_e\}$ and using the location axiom which states that $\bigvee_\ell at\ell = \underline{\text{true}}$, we get:

$$\vdash Q(\bar{y}) \supset \bigcirc Q(\bar{y}) .$$

Hence by Generalization (R3)

$$\vdash \Box(Q(\bar{y}) \supset \bigcirc Q(\bar{y})) .$$

By the induction axiom C8 we get:

$$\vdash Q(\bar{y}) \supset \Box Q(\bar{y}) .$$

Consider now an initial state at which we have $at\ell_0$ and φ true. By (a) φ implies Q , from which we conclude

$$\vdash [at\ell_0 \wedge \varphi] \supset \Box Q(\bar{y}) ,$$

which establishes the invariance of Q .

ACKNOWLEDGEMENTS: We wish to thank Bill Scherlis for his critical reading of the manuscript.

RELATED REFERENCES:

[BUR] Burstall, R.M. "Formal Description of Program Structure and Semantics of First-Order Logic", in Machine Intelligence 5, B. Meltzer and D. Michie (eds.), Edinburgh Press, pp. 79-98 (1970).

[CON] Constable, R.L. "On the Theory of Programming Logic", Proceedings of the 9th Annual Symposium on Theory of Computing, Boulder, Colorado (May 1977).

[FLO] Floyd, R. W., "Assigning Meanings to Programs", Proc. Symp. Appl. Math. 19, in J.T. Schwartz (ed.), Mathematical Aspects of Computer Science, American Mathematical Society, Providence, R.I. (1967), pp. 19-32.

[HAR] Harel, D. "Logic of Programs: Axiomatic and Descriptive Power", Ph.D. Thesis, Laboratory of Computer Science, M.I.T. (May 1978).

[HOA] Hoare, C.A.R. "An Axiomatic Basis of Computer Programming", CACM, Vol. 12, No. 10 (October 1969).

[H&C] Hughes, G.E. and Cresswell, M.J. "An Introduction to Modal Logic", Methuess & Co., London (1968).

[M&W] Manna, Z. and Waldinger, R. "Is 'Sometime' Sometimes Better than 'Always'?: Intermittent Assertions in Proving Program Correctness", CACM, Vol. 21, No. 2, pp. 159-172 (February 1978).

[PNU] Pnueli, A. "The Temporal Semantics of Concurrent Programs", Technical Report, Tel-Aviv University (1978).