RAIRO. INFORMATIQUE THÉORIQUE

F. KRÖGER

On temporal program verification rules

RAIRO. Informatique théorique, tome 19, n° 3 (1985), p. 261-280

<http://www.numdam.org/item?id=ITA_1985__19_3_261_0>

R.A.I.R.O. Informatique théorique/Theoretical Informatics
(vol. 19, n° 3, 1985, p. 261 à 280)

ON TEMPORAL PROGRAM VERIFICATION RULES (*)

by F. KRÖGER (1)

Communicated by K. APT

Abstract. — *This paper suggests a slight extension of the usual temporal logical framework for the description and verification of programs. With this extension it is possible to give elegant and transparent formulations of proof rules for formulas expressing program properties. Besides the transcription of well-known rules the paper particularly deals with formulas containing the recently introduced* atnext *operator.*

Résumé. — *Cet article présente une légère extension de la logique temporelle classique destinée à la description et la vérification des programmes. Muni de cette extension il est possible de donner des formulations élégantes et transparentes des formules exprimant les propriétés des programmes. En dehors de la transcription de règles bien connues, l'article étudie plus particulièrement les formules contenant l'opérateur récemment introduit « atnext ».*

1. INTRODUCTION

In the last few years temporal logic has been developed to an elegant and powerful tool for describing and proving properties of sequential and — particularly — parallel programs. In a series of papers [3, 4, 5], Manna and Pnueli have established a stock of useful proof rules concerning various kinds of program properties. Consider, for example, the " invariance rule " [5, with slightly changed notation] :

$$\frac{\begin{array}{l}\vdash \text{start} \rightarrow A \\ \vdash \text{Every transition of the program leads from } A \text{ to } A \\ \vdash A \rightarrow B\end{array}}{\vdash \Box B\,.}$$

(*) Received in March 1984, revised in September 1984.
(1) Institut für Informatik der Technischen Universität München, D-8000 München 2, Postfach 20 2420.

The second premise of this rule is a somewhat informal abbreviation for a set of premises each of which is a first-order formula expressing the complete " state-transition " function of a single transition of the program.

In this paper, we suggest to use *temporal* logical means to express such premises (which occur in all rules similarly). This leads — as we think — to more elegant (formal) versions of the rules. Moreover, and more interestingly, we then are able to

— express and justify the rules completely formally without referring to (and hence even without already having specified) the concrete effect of a transition as it is (implicitly) done in the above approach. This supports a more flexible use of the logical framework : The specification of different aspects of transitions can very easily be separated and the formal incorporation of other proved or presupposed assertions into an actual program proof is simplified.

To achieve this goal we slightly extend the classical way of temporal description of programs as given, e.g., in [3].

Besides the transcription of well-known rules mainly for *invariance* and *liveness* formulas we particularly want to deal with the temporal operator atnext introduced in [2]. We give basic proof rules for this operator and its iterations and indicate how it can be applied to describe program properties.

2. TEMPORAL LOGIC

We first establish the pure temporal logical framework on which we want to base description and verification of programs. The " classical " temporal system uses the *nexttime* operator $\bigcirc$, the *henceforth* or *always* operator $\Box$, the *sometime* operator $\Diamond$ and the (" strong ") *until* operator until [3]. It has been argued recently [2, 5] that instead of until one should take a " weak " operator (without any " existential " aspect) like the weak until operator unless [5] or the *first time* operator atnext [2]. We here take the latter one since we also want to give examples how to use just this operator.

Thus, let $\mathscr{L}$ be a first-order language (with equality) with operators $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\forall$ and the additional grammatical rules that

$$\bigcirc A,\ \Box A,\ A \text{ atnext } B$$

are formulas if A and B are. ($\Diamond A$ can be defined to be $\neg \Box \neg A$, and we also assume $\exists x A$ to be defined by $\neg \forall x \neg A$.) The informal meaning of A atnext B is : " A will be true at the next time point that B is true " (not assuming that B will be true at all).

Formal semantics of $\mathscr{L}$ can be defined as usual by the concept of a *Kripke structure* $\mathbb{K}$ consisting of a denumerable sequence $\{ \eta_0, \eta_1, \eta_2, \ldots \}$ of *states*. Every state η_i associates a truth value $\eta_i(A) \in \{ \mathbf{t}, \mathbf{f} \}$ with every formula A of $\mathscr{L}$ (" truth value of A in state η_i "). We only note the relevant rules for the temporal operators :

- $\eta_i(\bigcirc A) = \mathbf{t}$ iff $\eta_{i+1}(A) = \mathbf{t}$,
- $\eta_i(\Box A) = \mathbf{t}$ iff $\eta_j(A) = \mathbf{t}$ for all $j \geqslant i$,
- $\eta_i(A \underline{\text{ atnext }} B) = \mathbf{t}$ iff $\eta_j(B) = \mathbf{f}$ for all $j > i$ or $\eta_k(A) = \mathbf{t}$ for the smallest $k > i$ with $\eta_k(B) = \mathbf{t}$.

For notational simplicity we establish a priority order $\neg$, $\bigcirc$, $\Box$, $\Diamond$, $\underline{\text{atnext}}$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$ of the operators with $\neg$ binding most and $\leftrightarrow$ binding least.

We next give a formal proof system Σ for this logic :

Axioms :

(ax1) All instances of tautologies of usual propositional logic

(ax2) $\neg \bigcirc A \leftrightarrow \bigcirc \neg A$

(ax3) $\bigcirc(A \rightarrow B) \rightarrow (\bigcirc A \rightarrow \bigcirc B)$

(ax4) $\Box A \rightarrow A \wedge \bigcirc \Box A$

(ax5) $\bigcirc \Box \neg B \rightarrow A \underline{\text{ atnext }} B$

(ax6) $A \underline{\text{ atnext }} B \leftrightarrow \bigcirc(B \rightarrow A) \wedge \bigcirc(\neg B \rightarrow A \underline{\text{ atnext }} B)$

(ax7) $\forall x A \rightarrow A_x(t)$

(ax8) $\forall x \bigcirc A \rightarrow \bigcirc \forall x A$

(ax9) $x = x$

(ax10) $x = y \wedge A \rightarrow A_x(y)$ if A contains no temporal operators.

In (ax7), $A_x(t)$ denotes the result of replacing the free occurrences of the *subject variable* x by the *term* t. (It is assumed that t contains no subject variable which is bound by a quantifier in A.) $A_x(y)$ in (ax10) is analogous.

Rules :

(mp) $A, A \rightarrow B \vdash B$

(gen) $A \rightarrow B \vdash A \rightarrow \forall x B$ (x not free in A)

(nex) $A \vdash \bigcirc A$

(ind) $A \rightarrow B, A \rightarrow \bigcirc A \vdash A \rightarrow \Box B$

Σ can easily be proved to be sound with respect to a fully elaborated formal

semantics along the lines indicated above. At least the propositional part of Σ (consisting of (ax1)-(ax6) and (mp), (nex) and (ind)) can be shown to be also complete with respect to the propositional " sublogic " of $\mathscr{L}$ [8].

In [3] and [2] extensive lists of formulas which are derivable in Σ can be found. We only note one such formula for later use :

$$(\bigcirc\wedge) \qquad \bigcirc(A \wedge B) \leftrightarrow \bigcirc A \wedge \bigcirc B .$$

Furthermore, we observe that, of course, classical propositional and first-order predicate logic are contained in $\mathscr{L}$ in the sense that classical rules like

$$A \to B, B \to C \vdash A \to C ,$$

$$A \vdash \forall x A , \quad \text{etc.} ,$$

are also applicable in $\mathscr{L}$. We will indicate the use of such rules by (prop) and (pred), respectively.

Let us now discuss how to derive some special kinds of formulas in Σ, viz. formulas of the form

$$A \to \Box B ,$$
$$A \to \Diamond B ,$$
$$A \to B \underline{\text{atnext}}\ C .$$

The simplest case is the one of formulas $A \to \Box B$. Σ already contains the *induction rule* (ind) for proving such formulas, and we only want to give a slight modification of this rule which will be more appropriate for later use and is easily derived from (ind) :

$$(\text{ind}') \qquad A \to B, B \to \bigcirc B \vdash A \to \Box B .$$

Only a little bit less obvious is the case of formulas $A \to B \underline{\text{atnext}}\ C$. In Σ, the atnext operator occurs in (ax5) and (ax6) which, however, are not very helpful for proving such formulas since (ax5) concerns only a trivial case and (ax6) is some kind of " recursive characterization " of $A \underline{\text{atnext}}\ B$. But this recursion can immediately be transformed to another induction rule :

$$(\text{indatnext}) \quad A \to \bigcirc(C \to B) \wedge \bigcirc(\neg C \to A) \vdash A \to B \underline{\text{atnext}}\ C .$$

Before turning to the more complicated case of formulas $A \to \Diamond B$, we also treat some other derived operators. Fore some applications it is interesting to consider the *iterated atnext* operator inductively defined by

$$A \underline{\text{atnext}}^1 B \equiv A \underline{\text{atnext}} B,$$

$$A \underline{\text{atnext}}^{n+1} B \equiv (A \underline{\text{atnext}}^n B) \underline{\text{atnext}} B.$$

For example, $A \underline{\text{atnext}}^2 B$ is $(A \underline{\text{atnext}} B) \underline{\text{atnext}} B$ and means informally

" A will be true the second time that B is true ".

For proving formulas of the form $A \to B \underline{\text{atnext}}^n C$ we have the following simple extension of (indatnext) :

(indatnext^n)
$$\begin{aligned} & A \to \bigcirc(C \to B_1) \wedge \bigcirc(\neg C \to A), \\ & B_1 \to \bigcirc(C \to B_2) \wedge \bigcirc(\neg C \to B_1), \\ & \vdots \\ & B_{n-1} \to \bigcirc(C \to B) \wedge \bigcirc(\neg C \to B_{n-1}) \\ \vdash\ & A \to B \underline{\text{atnext}}^n C. \end{aligned}$$

Another useful operator (as already mentioned) is the " weak until operator " unless (this denotation is taken from [5]). A formula A unless B informally means :

" A will be true between now and the time point when B will be true (if this happens at all ; if not, A is true forever). "

unless can be expressed by atnext in the following way :

$$A \underline{\text{unless}} B \equiv B \underline{\text{atnext}} (A \to B).$$

(Because of the well-known expressive power of unless there is, of course, also a converse transcription :

$$A \underline{\text{atnext}} B \equiv \neg B \underline{\text{unless}} (A \wedge B).)$$

Applying the above definition of unless in (indatnext) we get another induction rule for this operator which was already mentioned by Wolper [9] :

(indunless) $$A \to \bigcirc C \vee \bigcirc(A \wedge B) \vdash A \to B \underline{\text{unless}} C.$$

There can be defined many other similar operators. As a last example we take the *precedence* operator which we denote by before. A formula A before B informally means :

" If B will be true sometime then A will be true before ".

It can be defined by

$$A \underline{\text{ before }} B \equiv \neg B \underline{\text{ atnext }} (A \vee B)$$

and has the following induction rule :

(indbefore) $\quad A \to \bigcirc \neg C \wedge \bigcirc(A \vee B) \vdash A \to B \underline{\text{ before }} C.$

All these operators considered up to now have in common that they are " weak " in the sense that they do not express any existential quantification over time points. This is the reason why they all possess some characteristic *induction* principle. Note, by the way, that these rules are *propositional* rules which are derivable in the propositional kernel of our temporal logic.

The situation changes if we now consider formulas of the kind

$$A \to \Diamond B.$$

There is no propositional induction principle for such formulas in the above sense. The only systematic approach seems to be the " method of well-founded ordering " which can be viewed as an induction principle over " data " represented by subject variables in the formula and therefore refers inherently to the full first-order logic.

Suppose $\mathscr{L}$ to contain a special binary predicate symbol $\preccurlyeq$ and (for simplicity) a special subset of subject variables $z, z', z_1, z_2, \ldots$ on which $\preccurlyeq$ can be applied (i.e., $z \preccurlyeq z'$, etc., are formulas of $\mathscr{L}$; $z \prec z'$ is defined to be $z \preccurlyeq z' \wedge z \neq z'$). Let these variables range over a set Z and let $\preccurlyeq$ be semantically interpreted by a *well-founded* ordering on Z (i.e., a partial ordering without infinite decreasing sequences). We denote $\mathscr{L}$ in this case by $\mathscr{L}_{wf}$. If follows from the assumptions that in $\mathscr{L}_{wf}$ the following *transfinite induction axiom* holds :

(ti) $\quad \forall z(\forall z' \prec z \to A(z')) \to A(z)) \to A(z)$

where $A(z)$ denotes a formula A of $\mathscr{L}_{wf}$ containing z, $A(z')$ means $A_z(z')$. Additionally assuming that the variables $z, z', \ldots$ do " not change their values during time " which is formally expressed by the axiom

(z) $\quad B \to \bigcirc B$, if B contains no other variables than $z, z', \ldots$

one can prove for such a language $\mathscr{L}_{wf}$ a basic proof principle for formulas of the kind $A \to \Diamond B$:

(wfo) $\quad A(z) \to \Diamond (B \vee \exists z'(z' \prec z \wedge A(z')))$

$$\vdash \exists z A(z) \to \Diamond B, \text{ if } B \text{ does not contain } z.$$

It should be noted that the premise in (wfo) is itself a formula of the form $A \rightarrow \Diamond B$, so this rule cannot be used without any further means. Trivial proof rules to which this problem can often be reduced are

$$A \rightarrow B \vdash A \rightarrow \Diamond B,$$

$$A \rightarrow \bigcirc B \vdash A \rightarrow \Diamond B,$$

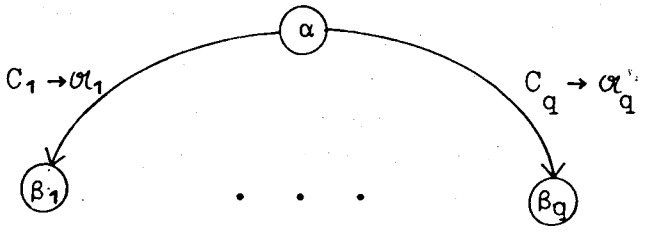
which follow immediately from (ax4).

3. PROGRAMS

We want to consider — as in [3, 4, 5] — programs of the form

initial R;
cobegin $\Pi_1 \parallel \ldots \parallel \Pi_p$ coend

where R is some *initial condition* and each *parallel component* Π_i is a sequential program. (Note that in the case $p = 1$ the whole program is sequential.) Π_i can be described by a transition graph a general cut-out of which looks like



The nodes $\alpha, \beta_1, \beta_2, \ldots$ of the graph correspond to a unique labeling of all instructions of Π_i. C_j is the *enabling condition* of the transition leading from α to β_j which must be true when this transition is to be executed. α_j denotes some " action " (e.g., an assignment $y := y + 1$) which is the effect of this transition. The formula $E_\alpha \equiv C_1 \vee \ldots \vee C_q$ is the *full exist condition* of α. The parallel execution of $\Pi_1, \ldots, \Pi_p$ is modelled by interleaving of transitions. (For more details, *cf.* [3].) The set of all labels (nodes) of such a program Π will be denoted by $\mathscr{M}_\Pi$ (analogously for Π_i, $1 \leqslant i \leqslant p$). A *program state* of Π is a $(p + 2)$-tuple $\eta = (\mu, \lambda_1, \ldots, \lambda_p, \kappa)$ where

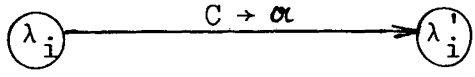
— μ assigns a value to each variable (" memory state "),
— $\lambda_i \in \mathscr{M}_{\Pi_i}$ for $i = 1, \ldots, p$,
— $\kappa \in \{ 0, 1, \ldots, p \}$.

η describes that each parallel component Π_i is ready to execute node λ_i

and that λ_κ is the next node which is actually executed (if $\kappa \neq 0$). $\kappa = 0$ means that no transition is executed (e.g., in the case of a deadlock). An *execution sequence* of Π is an infinite sequence $W_\Pi = \{ \eta_0, \eta_1, \eta_2, \ldots \}$ of program states with the following properties :

— $\eta_0 = (\mu_0, \alpha_0^{(1)}, \ldots, \alpha_0^{(p)}, \kappa_0)$ and R is true under μ_0. ($\alpha_0^{(i)}$ are the *initial labels* of Π_i.)

— If $\eta_j = (\mu, \lambda_1, \ldots, \lambda_i, \ldots, \lambda_p, i)$ then $\eta_{j+1} = (\mu', \lambda_1, \ldots, \lambda_i', \ldots, \lambda_p, \kappa')$, μ' is the new memory state resulting from executing α in μ, Π_i contains a transition



and C is true under μ.

— If $\eta_j = (\mu, \lambda_1, \ldots, \lambda_p, 0)$ then $\eta_{j+1} = \eta_j$ and E_{λ_i} is false under μ for every $i = 1, \ldots, p$.

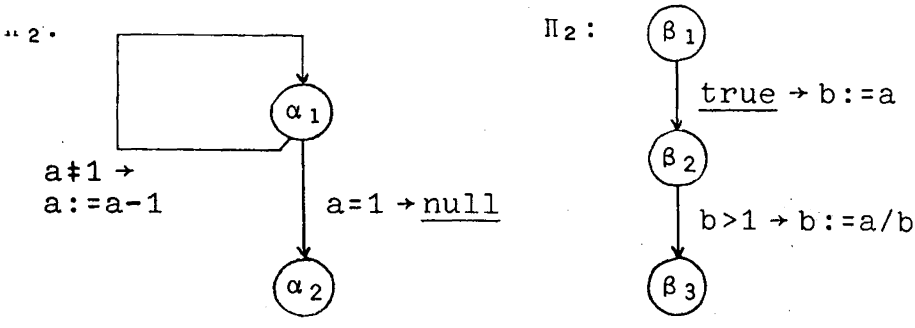
— If there are infinitely many $\eta_k = (\mu_k, \ldots)$ in W_Π containing some λ_i such that E_{λ_i} is true under μ_k then $\kappa = i$ in infinitely many of these η_k (*fair scheduling assumption*).

Let now $\mathscr{L}_\Pi$ be a language of temporal logic as described in Section 2 with the additional feature that for every $\alpha \in \mathscr{M}_\Pi$, α and atα are particular atomic formulas of $\mathscr{L}_\Pi$. The informal meaning of these formulas is :

α : " α is executed (next) "
atα : " α is ready to execute " .

atα is the usual kind of formula used in this context. We have argued in [1], that formulas of kind α are useful for different purposes, here we will use them particularly for formulating proof rules.

Let us give a little example to illustrate these notions. Consider a program with the initial condition $R \equiv a = 3$ and the parallel components



The *terminal* labels α_2 and β_3 can be viewed as having the full exit condition false. One possible execution sequence of this program is given by :

$$\eta_0 = (\mu_0, \alpha_1, \beta_1, 2) \quad \text{with} \quad \mu_0(a) = 3\,,$$
$$\eta_1 = (\mu_1, \alpha_1, \beta_2, 1) \quad \text{with} \quad \mu_1(a) = 3\,, \quad \mu_1(b) = 3\,,$$
$$\eta_2 = (\mu_2, \alpha_1, \beta_2, 1) \quad \text{with} \quad \mu_2(a) = 2\,, \quad \mu_2(b) = 3\,,$$
$$\eta_3 = (\mu_3, \alpha_1, \beta_2, 2) \quad \text{with} \quad \mu_3(a) = 1\,, \quad \mu_3(b) = 3\,,$$
$$\eta_4 = (\mu_4, \alpha_1, \beta_3, 1) \quad \text{with} \quad \mu_4(a) = 1\,, \quad \mu_4(b) = 1/3\,,$$
$$\eta_5 = (\mu_5, \alpha_2, \beta_3, 0) \quad \text{with} \quad \mu_5(a) = 1\,, \quad \mu_5(b) = 1/3\,,$$
$$\eta_6 = \eta_5\,,$$
$$\eta_7 = \eta_5\,,$$
$$\text{etc}\,.$$

The entry " 2 " in η_0 means that in the initial state where control is at α_1 and β_1, resp., Π_2 (i.e., β_1) is executed. Formally this will be mirrored by the formulas $at\alpha_1$, $at\beta_1$, and β_1 being true in η_0. In η_1 then control is at α_1 and β_2, resp., and Π_1 (i.e., α_1) executes (hence $at\alpha_1$, $at\beta_1$, α_1 are true) and this goes up to η_5 where α_2 and β_3 are reached and hence no action is executed anymore. All subsequent states remain unchanged.

We are interested now in Π-*valid* formulas, i.e., formulas which are valid in those Kripke structures where the sequences $\{ \eta_0, \eta_1, \eta_2, \ldots \}$ of states are execution sequences W_Π of Π. As indicated by the example, the semantics of the new kind of formulas is formally defined over such structures as follows :

$$\eta_i(\alpha) = \mathbf{t} \quad \text{iff} \quad \eta_i = (\mu, \lambda_1, \ldots, \lambda_p, \kappa) \quad \text{and} \quad \alpha = \lambda_\kappa\,,$$
$$\eta_i(at\alpha) = \mathbf{t} \quad \text{iff} \quad \eta_i = (\mu, \lambda_1, \ldots, \lambda_p, \kappa) \quad \text{and} \quad \alpha = \lambda_j \quad \text{for some } j = 1, \ldots, p.$$

Π-valid formulas can be derived by using the formal system Σ supplied with some additional axioms and one more rule describing just the restriction to execution sequences. We want to point out that these rule and axioms may be divided into three categories :

— the *basic rule* and two *basic axioms* which give only minimal information about the actual structure of the program but suffice to derive general proof rules for all kinds of program properties,
— *structural axioms* which hold for every program of the investigated class,
— *specification axioms* which specify the actual execution structure and the single actions of the program.

In order to formulate rule and axioms we introduce some notation : Let $\mathcal{M}_\Pi = \{ \alpha_1, ..., \alpha_l \}$ and $\mathcal{M}_{\Pi_i} = \{ \alpha_0^{(i)}, ..., \alpha_{m_i}^{(i)} \}$. We write

start_Π for $\text{at}\alpha_0^{(1)} \wedge ... \wedge \text{at}\alpha_0^{(p)} \wedge R$
(" the system is in its initial state "),

nil_Π for $\neg\alpha_1 \wedge \neg\alpha_2 \wedge ... \wedge \neg\alpha_l$
(" no action is executed ").

α, α', β, β_1, β_2... will be used as metavariables for elements of $\mathcal{M}_\Pi$.

Basic rule and axioms

(B1) $\quad \text{start}_\Pi \rightarrow \Box A \vdash A$

(B2) $\quad \text{nil}_\Pi \wedge A \rightarrow \bigcirc(\text{nil}_\Pi \wedge A)$

(B3) $\quad \Box\Diamond(\text{at}\alpha \wedge E_\alpha) \rightarrow \Diamond\alpha$

(B1) formalizes the fact that every execution sequence starts with a state in which start_Π is true. It is only another formulation of the rule (INIT) in [5]. It is remarkable that (B1) must really be given in the form of a rule. It is not possible to describe the same effect by axioms. (B2) expresses that " if no action is executed then nothing changes ". (B3) formalizes the fair scheduling assumption : " If α is enabled infinitely often then it is executed sometime ". (Note that we do not deal with *justice* conditions [5] in this paper. It is very easy to give an axiom like (B3) in order to restrict the considerations to *just* execution sequences.)

Structural axioms

(S1) $\quad \alpha \rightarrow \neg\alpha' \quad$ for $\quad \alpha \not\equiv \alpha'$

(S2) $\quad \alpha \rightarrow \text{at}\alpha$

(S3) $\quad \text{at}\alpha_j^{(i)} \rightarrow \neg\,\text{at}\alpha_k^{(i)} \quad$ for $\quad j \neq k$

(S4) $\quad \text{at}\alpha \wedge E_\alpha \rightarrow \neg\,\text{nil}_\Pi$

(S5) $\quad \text{at}\alpha \wedge \neg\alpha \rightarrow \bigcirc\text{at}\alpha$.

These axioms still do not describe the actual program but state general rules about execution sequences. Their informal meanings are :

(S1) : " No two actions are executed at the same time ".
(S2) : " α can only be executed if it is ready to ".
(S3) : " In every Π_i, no two actions are ready to execute at the same time ".
(S4) : " If some action is enabled then some action must execute ".
(S5) : " An action ready to execute but not executed remains ready ".

Specification axioms

The full specification of a concrete program finally consists of three parts :
— specification of the " computation structure " (the " topology " of the transition graph),
— specification of the data structure(s) envolved in the program,
— specification of the single actions.

The computation structure is specified by the following axiom :

(CS) $$\alpha \rightarrow (C_1 \wedge \bigcirc \mathrm{at}\beta_1) \vee \ldots \vee (C_q \wedge \bigcirc \mathrm{at}\beta_q) .$$

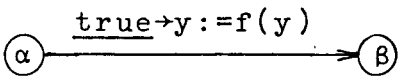
Here, α, $C_1, \ldots, C_q$, $\beta_1, \ldots, \beta_q$ are nodes and formulas as in the picture at the beginning of this section. (CS) describes the possible transitions from α to some β_i.

The data specification is carried out by first-order axioms and will not be followed up here. In applications we always will assume that appropriate axioms of this kind are given and indicate their use by (data).

The specification of a single action α can be carried out by giving some axiom of the form

$$\alpha \wedge A \rightarrow \bigcirc B \qquad (A, B \text{ usual first-order formulas}) .$$

It describes the effect of α on the program variables and can be compared with the Hoare's logic formula $A \{ \alpha \} B$. We only note, as an example, the case of an assignment :



The corresponding axiom is :

(assign) $$\alpha \wedge A_y(f(y)) \rightarrow \bigcirc A \qquad (A \text{ first-order formula}) .$$

(Note that the additional information $\alpha \rightarrow \bigcirc \mathrm{at}\beta$ is contained in (CS).) In applications we will indicate by (specΠ) the joint use of axioms (S1)-(S5), (CS) and axioms of this latter kind which, again, we assume implicitly given.

4. PROGRAM VERIFICATION PRINCIPLES

Many interesting properties of programs can be expressed by temporal formulas of the three kinds

$$A \to \Box B,$$
$$A \to \Diamond B,$$
$$A \to B \text{ atnext } C$$

(or, taking some other operator, $A \to B$ unless C, $A \to B$ before C, etc.).

We now give proof rules for such formulas in the context of some program Π. The justification of the rules is directly based on the respective logical rules noted in Section 2 and the basic program rule and axioms (B1)-(B3).

We begin, as an " auxiliary " step, with deriving a useful rule for proving a formula of the form $A \to \bigcirc B$.

(trans) $\quad \alpha \wedge A \to \bigcirc B$ for every $\alpha \in \mathscr{M}_\Pi$,
$\quad \text{nil}_\Pi \wedge A \to B$
$\quad \vdash A \to \bigcirc B$

(trans) is our formalization of the basic rule (TRNS) of [5].

Derivation (Let $\mathscr{M}_\Pi = \{ \alpha_1, ..., \alpha_n \}$) :

(1)	$\alpha \wedge A \to \bigcirc B$ for every $\alpha \in \mathscr{M}_\Pi$	assumption
(2)	$\text{nil}_\Pi \wedge A \to B$	assumption
(3)	$\text{nil}_\Pi \vee \alpha \vee ... \vee \alpha_n$	(ax1)
(4)	$\text{nil}_\Pi \wedge B \to \bigcirc B$	(B2), ($\bigcirc \wedge$), (prop)
(5)	$\text{nil}_\Pi \wedge A \to \bigcirc B$	(prop), (2) (4)
(6)	$A \to \bigcirc B$	(prop), (1), (3), (5).

The basic rule for proving formulas $A \to \Box B$ is the *invariance rule* :

(inv) $\quad A \to B,$
$\quad \alpha \wedge B \to \bigcirc B$ for every $\alpha \in \mathscr{M}_\Pi$
$\quad \vdash A \to \Box B$

Derivation :

(1)	$A \to B$	assumption

(2) $\alpha \wedge B \to \bigcirc B$ for every $\alpha \in \mathscr{M}_\Pi$ assumption

(3) $\mathrm{nil}_\Pi \wedge B \to B$ (ax1)

(4) $B \to \bigcirc B$ (trans), (2), (3)

(5) $A \to \Box B$ (ind'), (1), (4).

Note that we have not used (B1) and (B3) in this derivation but only (B2) because of (trans). (B1) can be used to derive a special case of (inv) where the conclusion is of the simpler form $\Box B$ (this rule corresponds to the rule (INV) in [5]) :

(inv') $\mathrm{start}_\Pi \to B$,
$\alpha \wedge B \to \bigcirc B$ for every $\alpha \in \mathscr{M}_\Pi$
$\vdash \Box B$

Next we give a rule for proving formulas $A \to \Diamond B$. We assume the underlying first-order language to be some $\mathscr{L}_{wf}$. Furthermore, we assume the existence of a *helpfulness* function $h : Z \to \{1, ..., p\}$, where Z is the well-founded range of the variables $z, z', ...$ Let $\mathscr{M}_{\Pi_{h(z)}} = \{\beta_1, ..., \beta_m\}$ and

$$E^{(h(z))} \equiv (\mathrm{at}\beta_1 \wedge E_{\beta_1}) \vee ... \vee (\mathrm{at}\,\beta_m \wedge E_{\beta_m}).$$

(well) $\alpha \wedge A(z) \to \bigcirc(B \vee \exists z'(z' \preccurlyeq z \wedge A(z')))$ for every $\alpha \in \mathscr{M}_\Pi \setminus \mathscr{M}_{\Pi_{h(z)}}$,
$\alpha \wedge A(z) \to \bigcirc(B \vee \exists z'(z' \prec z \wedge A(z')))$ for every $\alpha \in \mathscr{M}_{\Pi_{h(z)}}$,
$\Box A(z) \to \Diamond (B \vee E^{(h(z))})$
$\vdash \exists z A(z) \to \Diamond B$ (B not containing z).

The full derivation is quite clumsy, so we only note the main steps :

(1) $\alpha \wedge \exists z'(z' \preccurlyeq z \wedge A(z')) \wedge \Box \neg B \to \bigcirc(\exists z'(z' \preccurlyeq z \wedge A(z')) \wedge \Box \neg B)$ for all $\alpha \in \mathscr{M}_\Pi$ from the first two assumptions

(2) $A(z) \wedge \Box \neg B \to \Box \exists z'(z' \preccurlyeq z \wedge A(z'))$ from (1) with (inv), (prop), (pred)

(3) $A(z) \wedge \Box \neg B \wedge \Box \neg \exists z'(z' \prec z \wedge A(z')) \to \Diamond (\beta_1 \wedge A(z) \wedge \Box \neg B) \vee ...$
$... \vee \Diamond (\beta_m \wedge A(z) \wedge \Box \neg B)$ from (2) and the third assumption with (B3) and (pred)

(4) $A(z) \to \Diamond (B \vee \exists z'(z' \prec z \wedge A(z')))$ from (3) and the second assumption with (prop)

(5) $\exists z A(z) \to \Diamond B$ (wfo), (4)

(well) is a modification of the rule (WELL) of [5] which is formulated for just computations. Other rules like those in [4] could also be formulated and derived.

We turn to formulas of the kind $A \to B$ atnext C. A basic rule for such formulas is derived quite analogously to (inv) using (trans) and (indatnext) :

(atnext) $\alpha \wedge A \to \bigcirc(C \to B) \wedge \bigcirc(\neg C \to A)$ for every $\alpha \in \mathcal{M}_\Pi$,
$\text{nil}_\Pi \wedge C \to B$
$\vdash A \to B$ atnext C.

In [2], we have noted some useful special rules for this kind of formulas. These can be derived very easily from (atnext). We can also extend (atnext) to formulas with the iterated atnext operator :

(atnext^n) $\alpha \wedge A \to \bigcirc(C \to B_1) \wedge \bigcirc(\neg C \to A)$ for every $\alpha \in \mathcal{M}_\Pi$,
$\alpha \wedge B_1 \to \bigcirc(C \to B_2) \wedge \bigcirc(\neg C \to B_1)$ for every $\alpha \in \mathcal{M}_\Pi$,
$\vdots$
$\alpha \wedge B_{n-1} \to \bigcirc(C \to B) \wedge \bigcirc(\neg C \to B_{n-1})$ for every $\alpha \in \mathcal{M}_\Pi$,
$\text{nil}_\Pi \wedge C \to B_1 \wedge B_2 \wedge \ldots B_{n-1} \wedge B$
$\vdash A \to B$ atnext^n C.

We finally note rules for the unless and the before operator which can be based on (indunless) and (indbefore) :

(unless) $\alpha \wedge A \to \bigcirc C \wedge \bigcirc(A \wedge B)$ for every $\alpha \in \mathcal{M}_\Pi$,
$\text{nil}_\Pi \wedge A \to B \vee C$
$\vdash A \to B$ unless C.

In order to compare this rule with, say, the rule (CORE-U) of [5] we have to be a little bit careful because the unless operator is defined somewhat differently there. An appropriate transcription of that rule would be

$$\alpha \wedge A \to \bigcirc(A \vee B) \quad \text{for every} \quad \alpha \in \mathcal{M}_\Pi \vdash A \to A \text{ unless } B$$

and this follows immediately with (unless). We also could easily extend this rule for nested unless formulas as done in (CORE-U).

(before) $\alpha \wedge A \to \bigcirc \neg C \wedge \bigcirc(A \vee B)$ for every $\alpha \in \mathcal{M}_\Pi$,
$\text{nil}_\Pi \wedge A \to \neg C$
$\vdash A \to B$ before C.

5. APPLICATIONS OF THE ATNEXT OPERATOR

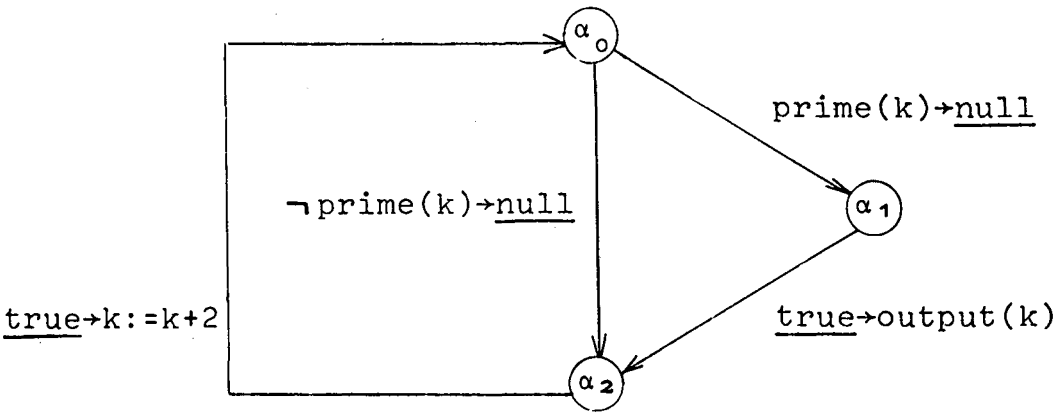
One typical application of the first time operator atnext is illustrated by the following simple sequential program.

Example 1

Π :

initial $k = 3$;
loop α_0 : if prime(k) then
α_1 : output(k) fi;
α_2 : $k := k + 2$
end

Π outputs all odd prime numbers in their natural order if we assume that prime(k) is a predicate which holds if and only if k is prime. The loop of Π can be described by the following transition graph :



If we now define the function nextprime by

$$\text{nextprime}(n) := \text{smallest prime number } m > n$$

then we are able to express the desired effect of Π by the following two formulas :

(C1) $\text{start}_\Pi \rightarrow k = 3$ atnext $\text{at}\alpha_1$
(C2) $\text{at}\alpha_1 \wedge k = k_0 \rightarrow k = \text{nextprime}(k_0)$ atnext $\text{at}\alpha_1$

(C1) expresses that the first number which is output is 3.
(C2) says that if some k_0 is output then the next output will be nextprime(k_0).

(C1) is trivial since we have $\text{start}_\Pi \to \bigcirc(\text{at}\alpha_1 \wedge k = 3)$. From this we get with (prop)

$$\text{start}_\Pi \to \bigcirc(\text{at}\alpha_1 \to k = 3) \wedge \bigcirc(\neg\, \text{at}\alpha_1 \to \text{start}_\Pi)$$

and by direct application of (indatnext) we get (C1).

In order to prove (C2) we first have to specify the actions null and output(k). (The specification of $k := k + 2$ falls under the scheme (assign) indicated in section 3.) null and also output(k) do not change any variable, thus we have

(1) $\alpha_0 \wedge P \to \bigcirc P$ for every first-order formula P

(2) $\alpha_1 \wedge P \to \bigcirc P$ for every first-order formula P.

Let now $A \equiv (\text{at}\alpha_0 \wedge k > k_0 \wedge k \leqslant \text{nextprime}(k_0)) \vee$
$(\text{at}\alpha_1 \wedge k = k_0) \vee$
$(\text{at}\alpha_2 \wedge k \geqslant k_0 \wedge k < \text{nextprime}(k_0))$.

We then have

(3)	$\alpha_0 \wedge A \wedge \text{prime}(k) \to \bigcirc(\text{at}\alpha_1 \wedge k = \text{nextprime}(k_0))$	(specΠ), (data)
(4)	$\alpha_0 \wedge A \wedge \neg\, \text{prime}(k) \to \bigcirc(\text{at}\alpha_2 \wedge k \geqslant k_0 \wedge k < \text{nextprime}(k_0))$	(specΠ), (data)
(5)	$\alpha_0 \wedge A \to \bigcirc(\text{at}\alpha_1 \to k = \text{nextprime}(k_0)) \wedge \bigcirc(\neg\, \text{at}\alpha_1 \to A)$	(prop), (3), (4)
(6)	$\alpha_1 \wedge A \to \bigcirc(\text{at}\alpha_1 \to k = \text{nextprime}(k_0)) \wedge \bigcirc(\neg\, \text{at}\alpha_1 \to A)$	similarly
(7)	$\alpha_2 \wedge A \to \bigcirc(\text{at}\alpha_1 \to k = \text{nextprime}(k_0)) \wedge \bigcirc(\neg\, \text{at}\alpha_1 \to A)$	similarly

If we now can show that

(8) $$\text{nil}_\Pi \wedge \text{at}\alpha_1 \to k = \text{nextprime}(k_0)$$

then (C2) follows by (atnext) from (5), (6), (7) and (8).

(8) follows directly from

(9) $$\Box \neg\, \text{nil}_\Pi$$

which is derived as follows :

(10)	$\text{start}_\Pi \to \text{at}\alpha_0 \wedge E_{\alpha_0}$	(prop)
(11)	$\text{start}_\Pi \to \neg\, \text{nil}_\Pi$	(prop), (S4)
(12)	$\alpha_0 \wedge \neg\, \text{nil}_\Pi \to \bigcirc(\alpha_1 \wedge E_{\alpha_1}) \vee \bigcirc(\alpha_2 \wedge E_{\alpha_2})$	(specΠ)
(13)	$\alpha_0 \wedge \neg\, \text{nil}_\Pi \to \bigcirc \neg\, \text{nil}_\Pi$	(S4)
(14)	$\alpha_1 \wedge \neg\, \text{nil}_\Pi \to \bigcirc \neg\, \text{nil}_\Pi$	similarly
(15)	$\alpha_2 \wedge \neg\, \text{nil}_\Pi \to \bigcirc \neg\, \text{nil}_\Pi$	similarly
(16)	$\Box \neg\, \text{nil}_\Pi$	(inv'), (11), (13), (14), (15).

A much more complicated and parallel example (the *alternating bit protocol*) with similar " correctness " assertions is treated in [2]. It should be noted that in this example it is also essential to use formulas of the kind α instead of atα for describing the desired properties.

Another field of application are general *precedence* properties which are expressed in the literature mostly with the until or unless operator (*cf.* [3, 5]). We explain this by an example taken from [5].

Example 2

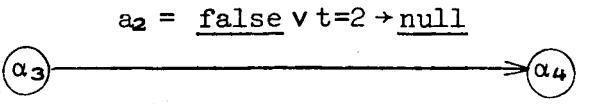
Π : initial $a_1 = \text{false} \wedge a_2 = \text{false} \wedge t = 1$;
cobegin $\Pi_1 \parallel \Pi_2$ coend

with

Π_1 : loop α_0 : .
.
.
α_1 : $a_1 := \text{true}$;
α_2 : $t := 1$;
α_3 : await $a_2 = \text{false} \vee t = 2$;
α_4 : .
. ⟨ critical section ⟩
.
α_5 : $a_1 := \text{false}$;
.
.
.
end

Π_2 : loop β_0 : .
.
.
β_1 : $a_2 := \text{true}$;
β_2 : $t := 2$;
β_3 : await $a_1 = \text{false} \vee t = 1$;
β_4 : .
. ⟨ critical section ⟩
.
β_5 : $a_2 := \text{false}$;
.
.
.
end

This program is a solution of the mutual exclusion problem [6]. It should be noted that the action α_3 corresponds to a transition of the form



and β_3 is analogous.

We assert 1-*bounded overtaking* which — by symmetry — has only to be expressed for one of the parallel components :

(C) $\text{at}\alpha_3 \rightarrow \text{at}\alpha_4 \text{ atnext}(\text{at}\alpha_4 \vee \text{at}\beta_4) \vee \text{at}\alpha_4 \text{ atnext}^2(\text{at}\alpha_4 \vee \text{at}\beta_4)$.

(C) says that if Π_1 is waiting for entering its critical section then Π_2 can enter its critical section before Π_1 at most once.

For proving (C) we first let L_1 and L_2 be the sets of labels between α_2 and α_5 (both included), and β_2 and β_5 (both included), respectively. If $L_i = \{\gamma_1, ..., \gamma_l\}$ we let $\mathrm{at}L_i \equiv \mathrm{at}\gamma_1 \vee ... \vee \mathrm{at}\gamma_l$.

Next we note some invariance properties of Π :

(1) $\quad \Box(t = 1 \vee t = 2)$

(2) $\quad \Box(a_1 = \underline{\mathrm{true}} \leftrightarrow \mathrm{at}L_1)$

(3) $\quad \Box(a_2 = \underline{\mathrm{true}} \leftrightarrow \mathrm{at}L_2)$

(4) $\quad \Box \neg \mathrm{nil}_\Pi$.

Everyone of these formulas can very easily be verified by using the invariance rule (inv'). Let now

$$A_1 \equiv \mathrm{at}\alpha_3 \wedge \mathrm{at}\beta_3 \wedge t = 1,$$
$$A_2 \equiv \mathrm{at}\alpha_3 \wedge (\mathrm{at}\beta_3 \rightarrow t = 2),$$
$$B \equiv \bigcirc(\mathrm{at}\alpha_4 \vee \mathrm{at}\beta_4 \rightarrow A_2) \wedge \bigcirc(\neg(\mathrm{at}\alpha_4 \vee \mathrm{at}\beta_4) \rightarrow A_1).$$

From (1)-(3) it is easy to derive

(5) $\quad \mathrm{at}\alpha_3 \rightarrow A_1 \vee A_2$

(6) $\quad \alpha \wedge A_1 \rightarrow B \quad$ for every $\alpha \in \mathscr{M}_\Pi$

(7) $\quad \alpha \wedge A_2 \rightarrow \bigcirc(\mathrm{at}\alpha_4 \vee \mathrm{at}\beta_4 \rightarrow \mathrm{at}\alpha_4) \wedge \bigcirc(\neg(\mathrm{at}\alpha_4 \vee \mathrm{at}\beta_4) \rightarrow A_2)$
for every $\alpha \in \mathscr{M}_\Pi$.

Because of (4) we need not care about the premises concerning nil_Π in the rules (atnext) and (atnext^2) and therefore we get

(8) $\quad A_2 \rightarrow \mathrm{at}\alpha_4 \, \underline{\mathrm{atnext}}(\mathrm{at}\alpha_4 \vee \mathrm{at}\beta_4)$

(9) $\quad A_1 \rightarrow \mathrm{at}\alpha_4 \, \underline{\mathrm{atnext}^2}(\mathrm{at}\alpha_4 \vee \mathrm{at}\beta_4)$

from (6) and (7). (C) then follows directly from (5), (8) and (9).

It should be noticed that the formulas A_1 and A_2 divide the possible situations very naturally into two cases : A_1 describes the case that both Π_1 and Π_2 are trying to enter their critical section and it's Π_2's turn. It is intuitively obvious and is also directly shown by the proof that only in this case overtaking takes place. The other case, expressed by A_2, is that Π_1 tries to enter its critical section and Π_2 is either not doing so or it's Π_1's turn.

Finally we note that we could also give one single rule for formulas of the form

$$A \to B \underline{\text{atnext}}\ C \vee B \underline{\text{atnext}}^2\ C \vee \ldots \vee B \underline{\text{atnext}}^n\ C$$

(according to the use of (atnext) and (atnext^2) above) by taking the same premises as in (atnext^n) but extending the conclusion in an obvious way :

$$\begin{array}{ll}
\alpha \wedge A \to \bigcirc(C \to B_1) \wedge \bigcirc(\neg C \to A) & \text{for every } \alpha \in \mathcal{M}_\Pi, \\
\alpha \wedge B_1 \to \bigcirc(C \to B_2) \wedge \bigcirc(\neg C \to B_1) & \text{for every } \alpha \in \mathcal{M}_\Pi, \\
\vdots & \\
\alpha \wedge B_{n-1} \to \bigcirc(C \to B) \wedge \bigcirc(\neg C \to B_{n-1}) & \text{for every } \alpha \in \mathcal{M}_\Pi, \\
\text{nil}_\Pi \wedge C \to B_1 \wedge B_2 \wedge \ldots \wedge B_{n-1} \wedge B & \\
\vdash A \vee B_1 \vee \ldots \vee B_{n-1} \to B \underline{\text{atnext}}\ C \vee \ldots \vee B \underline{\text{atnext}}^n\ C &
\end{array}$$

6. CONCLUDING REMARKS

The basic technical suggestion of this paper is a slight modification of the usual linguistic and semantical temporal framework for describing programs and their properties by introducing additional atomic formulas α (" action α is executed ") besides the formulas atα (" α is ready to execute "). We have argued in another paper [1] that it is not quite satisfactory if one tries to *express* α by the other linguistic means as it is done in [7]. This is the reason for introducing it as an independent element of the language.

With the help of this kind of formulas we have given completely formal (and nevertheless " readable ") formulations of basic proof rules for verification of program properties deriving these rules directly from their respective purely logical counterparts. Furthermore, we showed how the new kind of atomic formulas can be used for specifying the "flow of control" and the change of values of variables in a concrete program.

Finally, we have given some illustrations how the atnext operator can be used. A typical application of this operator is to describe the occurrence of some sequence $Q_1, Q_2, Q_3, \ldots$ of " events " (expressed by formulas) in the execution of a program. Furthermore it can also be used for simple description of other precedence properties which are usually expressed in the literature by the (weak) until operator.

REFERENCES

1. F. KRÖGER, *Some new aspects of the temporal logic of concurrent programs*, Technical University of Munich, Institute of Informatics, Report I8311 (1983).
2. F. KRÖGER, *A generalized nexttime operator*, JCSS *29*, 80-98 (1984).
3. Z. MANNA and A. PNUELI, *Verification of concurrent programs : The temporal framework*, in : The correctness problem in computer science (R. S. Boyer and J. S. Moore, eds.), International Lecture Series in Computer Science, Academic Press, London 1981.
4. Z. MANNA and A. PNUELI, *Verification of concurrent programs : Temporal proof principles*, in : Logics of programs, Proc. 1981, Springer LNCS *131*, 200-252 (1981).
5. Z. MANNA and A. PNUELI, *Proving precedence properties : The temporal way*, in : Proc. 10th ICALP, Springer LNCS *154*, 491-512 (1983).
6. G. L. PETERSON, *Myths about the mutual exclusion problem*, Information Processing Letters *12*, 115-116 (1981).
7. A. PNUELI, *The temporal semantics of concurrent programs*, Theor. Comp. Science *13*, 45-60 (1981).
8. H. SCHLINGLOFF, *Beweistheoretische Untersuchungen zur temporalen Logik*, Diploma thesis, Technical University of Munich, Institute of Informatics (1983).
9. P. WOLPER, *Temporal logic can be more expressive*, Proc. 22nd Symp. on Found. of Comp. Sci., Nashville, TN, 340-348 (1981).