AN AXIOMATIZATION OF THE INTERMITTENT ASSERTION METHOD USING TEMPORAL LOGIC
(extended abstract)

Krzysztof R. APT

LITP, Université Paris 7, 2 Place Jussieu, 75251 Paris, France

Carole DELPORTE

LCR-Thomson, Domaine de Corbeville, 91401 Orsay, France

Abstract. The intermittent assertion method proposed by Burstall [B] and subsequently popularized by Manna and Waldinger [MW] is axiomatized using a fragment of temporal logic. The proposed proof system allows to reason about while-programs. The proof system is proved to be arithmetically sound and complete in the sense of Harel [H]. The results of the paper generalize a corresponding result of Pnueli [P] proved for unstructured programs.

The system decomposes into two parts. The first part allows to prove *liveness* properties using as axioms theorems of the second part allowing to prove simple *safety* properties.

The completeness proof is constructive and provides a heuristic for proving specific liveness formulas.

1. INTRODUCTION

In 1977 Pnueli [P] introduced temporal logic as a tool for reasoning about sequential and concurrent programs. This approach received subsequently a lot of attention and since then several proof systems based on temporal were proposed. These proof systems allow to prove more complicated properties of concurrent programs than partial correctness or deadlock freedom (see e.g. [MP 1], [MP 2], [OL]).

However, most of these systems allow to reason about *unstructured* programs only. The only exception is the proof system of Owicki and Lamport [OL]. We find that in order to reason about structured programs a firm theoretical basis should be first established. In our opinion this was not done in [OL] where various obvious or less obvious axioms and proof rules are missing.

To clarify these issues we carry out our analysis in the framework of while-programs. Several (but not all) of the introduced axioms and proof rules are also valid in the case of parallel programs.

By $\mathcal{W}$ we denote the class of while-programs which is defined as usual. The programs from $\mathcal{W}$ use variables, expressions and Boolean expressions of the language L. They are denoted by the letters S,T.

We allow formulas of the form at S and after S for $S \in \mathcal{W}$. They are called *control formulas* and are denoted by the letter $\mathcal{C}$.

From assertions and control formulas we can built up certain formulas which will be called *mixed formulas*. They are of the form $\mathcal{C} \wedge p$. Mixed formulas are denoted by the letter μ.

The first subsystem discussed in section 4 allows two type of formulas : $\mathcal{C} \wedge p \supset \mathcal{C}' \wedge q$ and $\mathcal{C} \wedge p \rightsquigarrow \mathcal{C}' \wedge q$. If in the first type of a formula $\mathcal{C} \equiv \mathcal{C}'$ we omit $\mathcal{C}'$. We also omit all assertions of the form true. The formulas of the form $\mu_1 \rightsquigarrow \mu_2$ will be of main interest. We call them *liveness formulas*.

3.- SEMANTICS

To interpret the meaning of the formulas allowed in the proof system we provide an appropriate class of models for them. These models have to take into account the semantics of programs as the formulas refer directly to them. Therefore we define first the semantics of programs appropriate for our purposes. This semantics is a slight variant of the one introduced in [HP].

Let I be an interpretation of the assertion language L with a nonempty domain D. By an *assignation* we mean a function assigning to each variable x of L a value from the domain D. By a *state* we mean a pair which consists of a program $S \in \mathcal{W}$ or an empty program E and an assignation. We denote states by the letter s. If s is a state then by $\bar{s}$ we denote the assignation being its component. For a set C of states we define $\bar{C}$ to be the corresponding set of assignations: $\bar{C} = \{\bar{s} : s \in C\}$.

The value of a term t in an assignation $\bar{s}$ (written as $\bar{s}(t)$) and a truth of a formula p of L in an assignation $\bar{s}$ (written as $\models_I p(\bar{s})$) are defined as usual.

We define now a transition relation "$\rightarrow$" between states. Intuitively, for $s_0 = \langle S_0, \bar{s}_0 \rangle$ and $s_1 = \langle S_1, \bar{s}_1 \rangle$ $s_0 \rightarrow s_1$ means that one step in execution of S_0 in assignation $\bar{s}_0$ leads to assignation $\bar{s}_1$ with S_1 being the remainder of S_0 to be executed. If S_0 terminates in $\bar{s}_1$ then S_1 is empty, i.e. $S_1 \equiv E$. We assume that for any program $S: (E ; S) = (S ; E) = S$.

We define the relation $s \rightarrow s_1$ by the following clauses

i) $\langle x:=t, \bar{s} \rangle \rightarrow \langle E, \bar{s}_1 \rangle$

where $\bar{s}_1(y) = \bar{s}(y)$ for $y \not\equiv x$ and $\bar{s}_1(x) = \bar{s}(t)$

ii) If $\langle S, \bar{s} \rangle \rightarrow \langle S_1, \bar{s}_1 \rangle$ then for any program T $\langle S ; T, \bar{s} \rangle \rightarrow \langle S_1 ; T, \bar{s}_1 \rangle$

iii) $\langle \underline{\text{if}}\ b\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2\ \underline{\text{fi}}, \bar{s}\rangle \to \langle S_1, \bar{s}\rangle$ if $\models_I b(\bar{s})$

iv) $\langle \underline{\text{if}}\ b\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2\ \underline{\text{fi}}, \bar{s}\rangle \to \langle S_2, \bar{s}\rangle$ if $\not\models_I b(\bar{s})$

v) $\langle \underline{\text{while}}\ b\ \underline{\text{do}}\ S\ \underline{\text{od}}, \bar{s}\rangle \to \langle S\ ;\ \underline{\text{while}}\ b\ \underline{\text{do}}\ S\ \underline{\text{od}}, \bar{s}\rangle$ if $\models_I b(\bar{s})$

vi) $\langle \underline{\text{while}}\ b\ \underline{\text{do}}\ S\ \underline{\text{od}}, \bar{s}\rangle \to \langle E, \bar{s}\rangle$ if $\not\models_I b(\bar{s})$

vii) $\langle E, \bar{s}\rangle \to \langle E, \bar{s}\rangle$

Let $\to^*$ denote the transitive closure of $\to$.

Given now a program T by an *execution sequence* of T we mean a maximal sequence of states $s_0, s_1, \ldots$ such that $s_0 = \langle T, \bar{s}_0\rangle$ and for $i = 0,1,\ldots$ $s_i \to s_{i+1}$ holds. Clause vii) implies that each execution sequence is infinite. Execution sequences are denoted by the letters σ, τ . If $\sigma = s_0, s_1, \ldots$ then by definition $\sigma^i = s_i, s_{i+1}, \ldots$

For a program T we denote by Σ_T the set of all of its execution sequences closed under the truncation operation σ^i, i.e. $\sigma \in \Sigma_T$ implies that for any $i \geq 0$ $\sigma^i \in \Sigma_T$. Of course Σ_T depends on the interpretation I.

Having defined semantics of the programs we now define semantics of the control formulas.

Let S be a subprogram of T. We define

$\models_{T,I}$ at $S(s)$ iff $\exists\, \sigma \in \Sigma_T$ (s is an element of σ) and $s = \langle S; S', \bar{s}\rangle$ for some S' ;

$\models_{T,I}$ after $S(s)$ iff $\exists\, s_1\ \exists\, \sigma \in \Sigma_T$ (s, s_1 are element of σ)

$$\langle S; S', \bar{s}_1\rangle = s_1 \to^* s = \langle S', \bar{s}\rangle \text{ for some } S'$$

and if $S' \not\equiv E$ then for no s_2 such that $s_1 \to^* s_2 \to^* s$ $s_2 = \langle S', \bar{s}_2\rangle$.

Intuitively, $\models_{T,I}$ at $S(s)$ holds if s is a state in an execution sequence of T such that the subprogram S is just to be executed in s. And $\models_{T,I}$ after $S(s)$ holds if s is a state in an execution sequence of T such that the execution of the subprogram S has just terminated in s. (Note that our interpretation of after S, differs from that of [OL].) Of course the above definitions are not sufficiently precise as various occurrences of S and S' in σ do not need to correspond with the same program. To avoid the confusion we should actually assign to each subprogram of T a unique label. It is clear how to perform this process and we leave it to the reader.

Note that

$\models_{T,I}$ at $T(s)$ iff $s = \langle T, \bar{s}\rangle$ and

$\models_{T,I}$ after $T(s)$ iff $s = \langle E, \bar{s}\rangle$ and $\exists\, \sigma \in \Sigma_T$ (s is an element of σ).

The truth of assertions does not depend on the programs and we naturally put

$$\models_{T,I} p(s) \quad \text{iff} \quad \models_I p(\bar{s})$$

where p is a formula of L.

The truth of mixed formulas and formulas of the form $\mu \supset p$ and $C \supset C'$ is now defined in a natural way.

Finally we define the truth of liveness formulas. It depends on an execution sequence as it states a property of execution sequences and not states. We define

$$\models_{T,I} \mu_1 \rightsquigarrow \mu_2(\sigma) \quad \text{iff} \quad \sigma \in \Sigma_T \quad \text{and}$$

$$\forall_j \, [\models_{T,I} \mu_1(s_j) \Rightarrow \exists \, k \geq j \models_{T,I} \mu_2(s_k)]$$

where $\sigma = s_0, s_1, \ldots$

To make the definition of truth uniform for all types of formulas considered here we define

$$\models_{T,I} \varphi(\sigma) \quad \text{iff} \quad \models_{T,I} \varphi(s_0) \quad \text{where} \quad \sigma = s_0, s_1, \ldots$$

for all formulas φ whose definition of truth depended on a state only.

We now say that a formula φ of any type *is true with respect to* T *and* I, written as $\models_{T,I} \varphi$, if for all $\sigma \in \Sigma_T$ $\models_{T,I} \varphi(\sigma)$ holds.

This completes the definition of semantics.

4.- A SUBSYSTEM FOR PROVING LIVENESS FORMULAS

We present here the first part of the proof system called L which is designed to prove the liveness formulas from a certain set of hypotheses. The proof system L consists of two parts. The first part specifies how the control moves through the program. It is motivated by similar proof rules and axioms given in [L] and [OL]. The while rule shows how to prove the liveness properties of a while loop. It is an obvious adaptation of the rule given in [H] appropriate for proving the termination of while loops.

The second part axiomatizes the temporal operator " " and shows how to manipulate the liveness formulas.

The first part consists of the following axioms and rules :

ASSIGNMENT AXIOM

A1 : $\vdash_T$ at $S \wedge p[t/x]$ after $S \wedge p$
where $S \equiv x{:=}t$ is a subprogram of T.

Here as usual, $p[t/x]$ stands for the result of substituting t for the free occurrences of x in p.

CONCATENATION AXIOMS and RULE

Let $S \equiv S_1;S_2$ be a subprogram of T

A2 : $\vdash_T$ at $S \supset$ at S_1

A3 : $\vdash_T$ at $S_1 \supset$ at S

A4 : $\vdash_T$ after $S_1 \supset$ at S_2

A5 : $\vdash_T$ at $S_2 \supset$ after S_1

if S_2 is not a <u>while</u> construct

A6 : $\vdash_T$ after $S_2 \supset$ after S

A7 : $\vdash_T$ after $S \supset$ after S_2

R1 : concatenation rule

$$\frac{\vdash_T \text{ after } S_o \supset \neg p}{\vdash_T \text{ at } S_2 \wedge p \supset \text{ after } S_1}$$

where $S_2 \equiv$ <u>while</u> b <u>do</u> S_o <u>od</u>

SELECTION AXIOMS AND RULES

Let $S \equiv$ <u>if</u> b <u>then</u> S_1 <u>else</u> S_2 <u>fi</u> be a subprogram of T

A8 : $\vdash_T$ at $S \wedge b \wedge p \rightsquigarrow$ at $S_1 \wedge p$

A9 : $\vdash_T$ at $S \wedge \neg b \wedge p \rightsquigarrow$ at $S_2 \wedge p$

A10 : $\vdash_T$ after $S_1 \supset$ after S

A11 : $\vdash_T$ after $S_2 \supset$ after S

R2 : $$\frac{\vdash_T \text{ after } S_1 \supset \neg q}{\vdash_T \text{ after } S \wedge q \supset \text{ after } S_2}$$

R3 : $$\frac{\vdash_T \text{ after } S_2 \supset \neg q}{\vdash_T \text{ after } S \wedge q \supset \text{ after } S_1}$$

WHILE AXIOMS AND RULES

Let $S \equiv$ <u>while</u> b <u>do</u> S_o <u>od</u> be a subprogram of T.

A12 : $\vdash_T$ at $S \wedge b \wedge p \rightsquigarrow$ at $S_o \wedge p$

A13 : $\vdash_T$ at $S \wedge \neg b \wedge p \rightsquigarrow$ after $S \wedge p$

A14 : $\vdash_T$ after $S_o \supset$ at S

R4 : $$\frac{\vdash_T \text{at}^+S \supset \neg q}{\vdash_T \text{at } S \wedge q \supset \text{after } S_o}$$

The formula at^+S attempts to describe the fact that the control is at the beginning of S for the first time.

The form of at^+S depends on the direct context of the <u>while</u> loop S within T. It is defined as follows

If S appears in T in the form :

$S_1;S$ then $\text{at}^+S \equiv \text{after } S_1$,

$T_1 \equiv$ <u>if</u> b_1 <u>then</u> $S(;S_1)$ <u>else</u> S_2 <u>fi</u> then $\text{at}^+S \equiv \text{at } T_1 \wedge b_1$,

$T_1 \equiv$ <u>if</u> b_1 <u>then</u> S_1 <u>else</u> $S(;S_2)$ <u>fi</u> then $\text{at}^+S \equiv \text{at } T_1 \wedge \neg b_1$,

$T_1 \equiv$ <u>while</u> b_1 <u>do</u> $S(;S_1)$ <u>od</u> then $\text{at}^+S \equiv \text{at } T_1 \wedge b_1$.

If none of the above cases arises then T is of the form $S;S_1$ and we put $\text{at}^+S \equiv \text{at } T$.

R5 : <u>while</u>-rule

$$\frac{\vdash_T \text{at } S \wedge p(n+1) \supset b,\ \vdash_T \text{at } S_o \wedge p(n+1) \rightsquigarrow \text{after } S_o \wedge p(n)}{\vdash_T \text{at } S \wedge \exists\, n\ p(n) \rightsquigarrow \text{at } S \wedge p(0)}$$

Here $p(n)$ is an assertion with a free variable n which does not appear in S and ranges over natural numbers.

The second part of the system L consists of the following rules :

R6 : reflexivity rule

$$\frac{\vdash_T \mu_1 \supset \mu_2}{\vdash_T \mu_1 \rightsquigarrow \mu_2}$$

R7 : transitivity rule

$$\frac{\vdash_T \mu_1 \rightsquigarrow \mu_2,\ \vdash_T \mu_2 \rightsquigarrow \mu_3}{\vdash_T \mu_1 \rightsquigarrow \mu_3}$$

R8 : confluence rule

$$\frac{\vdash_T \mu_1 \wedge b \rightsquigarrow \mu_2,\ \vdash_T \mu_1 \wedge \neg b \rightsquigarrow \mu_2}{\vdash_T \mu_1 \rightsquigarrow \mu_2}$$

We also adopt without mentioning all axioms and proof rules of classical logic concerning $\supset$ and $\wedge$ applied to formulas $\mu_1 \supset \mu_2$ and their special cases.

The system L allows to prove $\vdash_T C \wedge p \supset C'$ whenever $\models_{T,I} C \wedge p \supset C'$. Thus if we wish to prove $\vdash_{T,I} C \wedge p \supset C' \wedge q$ it suffices to prove $\vdash_T C \wedge p \supset q$.

In section 7 we present another part of the proof system which allows to prove such formulas directly from assertions. For a moment we accept these formulas as axioms.

Let A be a set of the formulas of the form $\vdash_T \mu \supset p$. Given a liveness formula φ we say that φ *can be proved from* A, written as $A \vdash_T \varphi$, if there exists a proof of φ in the proof system L which uses some of the elements of A as axioms.

5.- SOUNDNESS

In order to prove soundness of the proof system L we should interpret the formulas in a model. However, not all models are appropriate here. The reason for it is that the while rule R5 refers to natural numbers. To ensure a correct interpretation of this rule we should restrict ourselves to models which contain natural numbers. This leads us to *arithmetical interpretations* defined in [H]. We recall the definition :

let L^+ be the minimal extension of L containing the language L_p of Peano arithmetic and a unary relation nat(x). Call an interpretation I of L^+ *arithmetical* if its domain includes the set of natural numbers, I provides the standard interpretation for L_p, and nat(x), is interpreted as the relation "to be a natural number". Additionally, we require that there exists a formula of L^+ which, when interpreted under I, provides the ability to encode finite sequences of elements from the domain of I into one element. (The last requirement is needed only for the completeness proof.) Our proof system is suitable only for assertion languages of the form L^+, and an expression such as p(n) is actually a shorthand for nat(n) $\wedge$ p(n).

Given now a program T and an arithmetical interpretation I denote by Th(T,I) the set of all formulas of the form $\vdash_T \mu \supset q$ for which $\models_{T,I} \mu \supset q$.

We have the following theorem :

SOUNDNESS THEOREM

Let T be a program from W and let I be an arithmetical interpretation. For any liveness formula φ if $Th(T,I) \vdash_T \varphi$ then $\models_{T,I} \varphi$. □

Note that any liveness formula true or provable in a context of T refers to subprograms of T only.

6.- COMPLETENESS

The following theorem states completeness of the subsystem L.

COMPLETENESS THEOREM

Let T be a program from $\mathbb{V}$ and let I be an arithmetical interpretation. For any liveness formula φ if $\models_{T,I} \varphi$ then $Th(T,I) \vdash_T \varphi$.

The proof of the theorem relies on the following important proposition.

Proposition 1 : Let S be a subprogram of T. Then for any liveness formula φ if $Th(S,I) \vdash_S \varphi$ then $Th(T,I) \vdash_T \varphi$.

Proof : The proof of $\vdash_S \varphi$ becomes a proof of $\vdash_T \varphi$ if we replace everywhere in it "$\vdash_S$" by "$\vdash_T$". □

This proposition has a semantic counterpart.

Proposition 2 : Let S be a subprogram of T. Then for any liveness formula φ if $\models_{S,I} \varphi$ then $\models_{T,I} \varphi$. □

The proof of the theorem proceeds by structural induction with respect to T. Given $\varphi \equiv \mu \rightsquigarrow \mu'$ we find in each case a chain of the intermediate mixed formulas $\mu_0, \mu_1, \ldots, \mu_k$ such that $\mu = \mu_0, \mu_k = \mu'$ and for each $i=0,\ldots,k-1$ $\models_T \mu_i \rightsquigarrow \mu_{i+1}$. This chain is so chosen that for every $i=0,\ldots,k-1$ either $\models_S \mu_i \rightsquigarrow \mu_{i+1}$ for a proper subprogram S of T or μ_i μ_{i+1} can be proved directly. In the first case by the induction hypothesis $Th(S,I) \vdash_S \mu_i \rightsquigarrow \mu_{i+1}$ so $Th(T,I) \vdash_T \mu_i \rightsquigarrow \mu_{i+1}$ by proposition 1. In the latter case one either applies the axioms or proof rules directly or makes use of the induction hypothesis. Depending on the case the length of the chain varies between 2 and 5. In some cases more than one chain is needed and the confluence rule is used to obtain the desired result.

7.- A SUBSYSTEM FOR PROVING FORMULAS OF THE FORM $\vdash_T \mu \supset p$

The subsystem presented in section 4 used as axioms formulas of the form $\vdash_T \mu \supset p$. Such a choice of axioms is unsatisfactory for our purposes as these formulas refer to programs and the properties expressed by them are not always easy to verify. Note for example that $\models_{T,I}$ after $T \wedge p \supset q$ is equivalent to $\models_{T,I} \{true\}\ T\ \{p \supset q\}$ in the sense of Hoare's logic (see e.g. [A]).

To remedy this deficiency we provide now another part of the proof system called $\mathcal{S}$ appropriate for proving this type of formulas.

The system $\mathcal{S}$ allows to prove arbitrary true formulas of the form $\vdash_T C \supset p$ so also $\vdash_T C \wedge p \supset q$ since $C \wedge p \supset q \equiv C \supset (p \supset q)$.

Two types of formulas are allowed in the system $\mathcal{S}$: $\mu \supset p$ and $C \supset C'$.

The system consists of the following axioms and rules :

ASSIGNMENT RULE

S1 : let $S \equiv x:=t$ be a subprogram of T

$$\frac{\vdash_T \text{at } S \supset p\,[t/x]}{\vdash_T \text{after } S \supset p}$$

SELECTION RULES

Let $S \equiv$ <u>if</u> b <u>then</u> S_1 <u>else</u> S_2 <u>fi</u> be a subprogram of T

S2 : $\dfrac{\vdash_T \text{at } S \supset p}{\vdash_T \text{at } S_1 \supset p \wedge b}$ if S_1 does not begin with a <u>while</u> loop

S3 : $\dfrac{\vdash_T \text{at } S \supset p}{\vdash_T \text{at } S_2 \supset p \wedge \neg b}$ if S_2 does not begin with a <u>while</u> loop

S4 : $\dfrac{\vdash_T \text{after } S_1 \supset p,\ \vdash_T \text{after } S_2 \supset r}{\vdash_T \text{after } S \supset p \vee r}$

CONCATENATION AXIOMS

Axioms A2 - A7

WHILE RULES

Let $S \equiv$ <u>while</u> b <u>do</u> S_o <u>od</u> be a subprogram of T

S5 : $\dfrac{\vdash_T \text{at } S \supset p}{\vdash_T \text{at } S_o \supset p \wedge b}$ if S_o does not begin with a <u>while</u> loop

S6 : $\dfrac{\vdash_T \text{at } S \supset p}{\vdash_T \text{after } S \supset p \wedge \neg b}$

S7 : $\dfrac{\vdash_T \text{at}^+ S \supset p,\quad \text{at } S_o \supset p \wedge b \vdash_T \text{after } S_o \supset p}{\vdash_T \text{at } S \supset p}$

The second premise of rule S7 means that there exists a proof of $\vdash_T$ after $S_o \supset p$ in the system from the assumption $\vdash_T$ at $S_o \supset p \wedge b$. This expresses in the system a property corresponding to $\{p \wedge b\}\ S_o\ \{p\}$ in the sense of Hoare's logic. Note that for any I : $\models_I \{p \wedge b\} S_o \{p\}$ implies $[\models_{T,I}$ at $S_o \supset p \wedge b \Rightarrow \models_{T,I}$ after $S_o \supset p]$ but not necessarily conversely. at$^+$S is defined in section 4.

INITIALIZATION AXIOM

B1 : $\vdash_T$ at T $\supset$ <u>true</u>.

Let A be a set of assertions. We say that a formula $\vdash_T C \supset p$ *can be proved from* A, written as $A \vdash_T C \supset p$, if there exists a proof in the above system which uses some of the elements of A as axioms.

We denote by Th(I) the set all assertions true in I.

The following theorem states arithmetical soundness and completeness of the system $\mathcal{S}$.

THEOREM Let T be a program from $\mathcal{W}$ and let I be an arithmetical interpretation. Then for any formula $C \supset p$

$$Th(I) \vdash_T C \supset p \text{ iff } \models_{T,I} C \supset p.$$

The completeness proof, i.e. the implication "$\Leftarrow$" proceeds by induction with respect to a certain well-ordering defined on the control formulas. This ordering is defined as follows. Consider the directed graph representing the flowchart of T with nodes being the control formulas. Remove now from this graph all edges causing cycles, i.e. edges leading from after S_0 to at S for any subprogram $S \equiv$ while b do S_0 od of T. The resulting graph defines the well-ordering in question. Due to the lack of space the details of the proof are omitted.

The converse implication, i.e. the soundness proof is straightforward. A precise proof requires techniques similar to those of section 3.7 of [A] to deal properly with rule S7.

COROLLARY Let T be a program from $\mathcal{W}$ and let I be an arithmetical interpretation. Then for any liveness formula φ

$$Th(I) \vdash_T \varphi \text{ iff } \models_{T,I} \varphi. \qquad \square$$

Here $\vdash_T$ refers to the provability in the final proof system which contains all mentioned axioms and rules.

Proofs will appear in the full version of the paper.

Acknowledgements. We are grateful to D. Lehmann for suggesting a simplified completeness proof of the system $\mathcal{L}$ and to E.-R. Olderog for critical remarks concerning the first version of the paper.

REFERENCES

[A] Apt, K.R., Ten Years of Hoare's logic, a survey, part I, TOPLAS, vol. 3,4, pp. 431-483, 1981.

[B] Burstall, R.M., Program proving as hand simulation with a little induction, in : Proceedings IFIP 74, pp. 308-312, North Holland, Amsterdam, 1974.

[H] Harel, D., First order dynamic logic, Lecture Notes in Computer Science, 68, Springer Verlag, 1979.

[HP] Hennessy, M.C.B., Plotkin G.D., Full abstraction for a simple programming language, in : Proceedings 8th Symposium MFCS, Lecture Notes in Computer Science, 74, pp. 108-120, 1979.

[Ho] Hoare, C.A.R., An axiomatic basis of computer programming, Communications ACM, vol. 12, 10, pp. 576-580, 583, 1969.

[L] Lamport, L., The "Hoare Logic" of concurrent programs, Acta Informatica, vol. 14, 1, pp. 21-37, 1980.

[MP1] Manna Z., Pnueli A., Verification of concurrent programs ; The temporal framework, in : The Correctness Problem in Computer Science, International Lecture Series in Computer Science, Academic Press, London, 1981.

[MP2] Manna Z., Pnueli A., Verification of concurrent programs ; Temporal proof principles, in : Logic of Programs, Lecture Notes in Computer Science, 131, pp. 200-252, 1982.

[MW] Manna Z., Waldinger R., Is "Sometime" sometimes better than "Always" ?, Communications ACM, vol. 21, 2, pp. 159-172, 1978.

[OL] Owicki S., Lamport L., Proving liveness properties of concurrent programs, TOPLAS, vol. 4, 3, pp. 455-495, 1982.

[P] Pnueli, A., The temporal logic of programs, in : Proceedings 18th Symposium FOCS, pp. 46-57, IEEE, Providence, R.I., 1977.