INFORMATION AND CONTROL **68**, 223–253 (1986)

Syntax Directed Analysis of Liveness Properties of While Programs

KRZYSZTOF R. APT AND CAROLE DELPORTE-GALLET

L.I.T.P., Université Paris 7, 2 Place Jussieu, 75251 Paris, France

A syntax directed proof system which allows to prove *liveness* properties of **while**-programs is introduced. The proof system is proved to be arithmetically sound and complete in the sense of Harel ("Lecture Notes in Comput. Sci. Vol. 68," Springer-Verlag, Berlin/New York, 1979). The results of the paper generalize a corresponding result Pnueli ("Proc. 18th Sympos. FOCS" IEEE, Providence, R. I., 1977) proves for unstructured programs. The proof system decomposes into two parts. The first part allows to prove simple *safety* properties. These properties are used as axioms in the second proof system which deals with *liveness* properties. The completeness proof is constructive and provides a heuristic for proving specific liveness properties.

Contents. 1. Introduction. 2. Preliminaries. 3. Semantics. 4. Expressiveness of the Syntax. 5. A Proof System $\mathscr{S}$ for Proving Formulas of the Form $|_T \mathscr{C} \supset p$. 6. Soundness and Arithmetical Completeness of $\mathscr{S}$. 7. A Proof System $\mathscr{L}$ for Proving Liveness Formulas. 8. Arithmetical Soundness and Completeness of $\mathscr{L}$. 9. Conclusions. References.

1. INTRODUCTION

In Pnueli (1977), temporal logic is as a tool for reasoning about sequential and concurrent programs. This approach subsequently received a lot of attention and since then several proof systems based on temporal logic were proposed. These proof systems allow us to prove more complicated properties of concurrent programs than partial correctness or deadlock freedom (see, e.g., Manna and Pnueli, 1981, 1982; Owicki and Lamport, 1982).

However, most of these systems allow us to reason about *unstructured* programs only. The only exception is the proof system of Owicki and Lamport (1982). We find that in order to reason about structured programs a firm theoretical basis should be first established. In our opinion this is not done in Owicki and Lamport (1982), where various obvious or less obvious axioms and proof rules are missing.

One of the most fundamental issues when reasoning about structured

0019-9958/86 $3.00

programs is the problem of specifying how the control moves throughout the program. Most of those issues are straightforward and often they do not depend on the fact of whether or not the program is parallel. On the other hand, the problem of *completeness* of these specifications is not obvious and requires a careful analysis.

We carry our our study in the framework of **while**-programs. Several (but not all) of the introduced axioms and proof rules are also sound in the case of parallel programs. A similar analysis in the case of parallel programs will necessarily depend on the results of this paper.

Temporal logic allows us to classify various program properties according to their syntactical form. The most important ones are those of *safety* (or *invariance*), and *liveness*. Safety properties (like partial correctness, deadlock freedom) are by now well understood and our paper concentrates on the issue of liveness properties. These are properties which assert that some desired event will eventually take place (e.g., termination or entering a critical section).

In the paper we provide a proof system which allows to prove arbitrary liveness properties of **while**-programs and prove its arithmetical soundness and completeness in the sense of Harel (1979). The results of the paper generalize a corresponding result of Pnueli (1977), proved for unstructured programs. Moreover, the proof system is *syntax directed*, i.e., the proof rules and axioms follow the syntax of the programs. The completeness proof is constructive and provides a heuristic for proving specific liveness formulas.

It is presented in Section 5 and proved sound and arithmetically complete in Section 6. This part of the proof system is partially motivated by Lamport (1980) and Owicki and Lamport (1982).

The second part of the system called $\mathscr{L}$, deals with liveness formulas. This subsystem is a mixture of axioms and proof rules motivated by Harel (1979); Hoare (1969); Lamport (1980); Owicki and Lamport (1982); and Pnueli (1977). It is presented in Section 7. The proof of arithmetical soundness and completeness of $\mathscr{L}$ *relative to* $\mathscr{S}$ is proved in Section 8.

Combining the proof systems $\mathscr{S}$ and $\mathscr{L}$ we get a hierarchically built proof system which allows to prove liveness properties directly from first-order assertions.

Finally, in Section 9 we present some conclusions and directions for the future work. An extended abstract of this paper appears as Apt and Delporte (1983).

Another approach to these issues has been recently proposed in Barringer, Kuiper, and Pnueli (1984) and Gerth (1984), where different formalisms are used. In these papers syntax directed proof systems for proving temporal properties of parallel programs are proposed.

We adopt here the formalism of Lamport (1980) and Owicki and Lam-

port (1982), where the formulas at S and after S for S being a program are introduced. The formula at S states that the subprogram S is about to be executed and the formula after S states that the execution of the subprogram S has just terminated. Our interpretation of after S differs from that of Owicki and Lamport (1982), for the reasons explained in Section 3.

In Section 2 we explain the notation and terminology used in the paper.

The basic constructs in our proof system are formulas of the form $\eta_0 S_0 \wedge p \rightsquigarrow \eta_1 S_1 \wedge q$, where $\eta_0, \eta_1 \in \{\text{at, after}\}$ which are called throughout the paper *liveness formulas*. The operator "$\rightsquigarrow$" is the "leads to" operator of temporal logic (see Pnueli, 1977 and Owicki and Lamport, 1982) and is interpreted as $p \rightsquigarrow q \equiv \Box(p \supset \Diamond q)$. Thus, $p \rightsquigarrow q$ represents a temporal implication or eventual occurence of q under the assumption of p.

To understand the essence of the problems investigated here let us consider the liveness formula $\varphi \equiv \text{at } S \wedge x = 0 \rightsquigarrow \text{after } S \wedge x = 5$, where $S \equiv x := x + 2$. This formula is of course false if we interpret it as $\{x = 0\}\ S\{x = 5\}$ in the sense of Hoare's logic (Hoare, 1969). However, if we consider S as a subprogram of the program $T \equiv x := 0;\ T'$, where $T' \equiv$ **while** $x < 10$ **do** $S;\ x := x + 1$ **od** then the formula φ is true. Thus the truth of the lieveness formulas depends on the *context* in which they are considered. We indicate this dependence by attaching the context program T to the truth relation "$\models$" and the provability relation "$\vdash$". In the course of the proofs (here of $\vdash_T \varphi$) we first prove the formulas in the minimal context in which they are true (here T') and subsequently extend the context to the desired one (here T).

While the interest in proving such formulas for **while**-programs is debatable, it should be noted that they naturally occur in the context of parallel programs. The problems we study in this paper should be in our opinion resolved first in the context of sequential problems before they can be considered in the context of parallel programs.

We introduce in Section 3 semantics of the concerning formulas and in Section 4 investigate what properties can be expressed using the syntax of the paper.

The proof system consists of two parts. The first of them called $\mathscr{S}$ is designed to prove simple safety properties which are needed in the proofs of liveness properties.

2. Preliminaries

As indicated in the introduction we are interested here in proving the formulas of the form $\eta_0 S_0 \wedge p \rightsquigarrow \eta_1 S_1 \wedge q$, where $\eta_0, \eta_1 \in \{\text{at, after}\}$, S_0, S_1 are **while**-programs and p, q are assertions. To this end we define various classes of formulas which will be used in the sequel.

Let L be a first-order language with equality. We call the formulas of L *assertions* and denote them by letters p, q, r. The letters x, y, z denote the variables of L, the letter t denotes the terms (*expressions*) of L, the letter b denotes quantifier-free formulas (*Boolean expressions*) of L.

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

In the sequel, we label each assignment within a given program by a unique label. In other words, a **while**-program is well formed if every assignment being its subprogram has a unique label attached to it. The use of labels allows us to distinguish between different occurences of the same subprogram in a given program.

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

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

In the proof system, formulas of the following types will be allowed: p, $\mu \supset p$, $\mu \supset \mathscr{C}$, $\mu_1 \supset \mu_2$ and $\mu_1 \rightsquigarrow \mu_2$. The formulas of the form $\mu \supset p$, $\mu \supset \mathscr{C}$ and $\mu_1 \supset \mu_2$ are called *safety formulas*. 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 Hennessy and Plotkin (1979).

First we introduce the notion of a *suffix* of a program. It is simply a part of its text which remains to be executed. In general, a suffix need not be a program—for example, S_1 **fi** is a suffix of **if** b **then** S_1 **else** S_2 **fi**. We denote suffices by the letter Δ and the empty suffix by the letter E.

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 suffix of a program from $\mathscr{W}$ 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 adopt here the point of view according to which very few control points in the program coincide. One step in execution of a program will consist either of executing an assignment statement or entering or leaving bodies of a **while** or **if** statement or passing to the next statement. Thus leaving an **if** statement will take one step.

We define now a transition relation "$\rightarrow$" between states. Intuitively, for $s_0 = \langle \Delta_0, \bar{s}_0 \rangle$ and $s_1 = \langle \Delta_1, \bar{s}_1 \rangle$ $s_0 \rightarrow s_1$ means that one step in execution of Δ_0 in assignation $\bar{s}_0$ leads to assignation $\bar{s}_1$ with Δ_1 being the remainder of Δ_0 to be executed. If Δ_0 terminates in $\bar{s}_1$ then Δ_1 is empty, i.e., $\Delta_1 \equiv E$.

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

(i) $\langle \alpha : x := t\Delta, \bar{s} \rangle \rightarrow \langle \Delta, \bar{s}_1 \rangle$, where $\bar{s}_1(y) = \bar{s}(y)$ for $y \not\equiv x$ and $\bar{s}_1(x) = \bar{s}(t)$,

(ii) $\langle \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi } \Delta, \bar{s} \rangle \rightarrow \langle S_1 \textbf{ fi } \Delta, \bar{s} \rangle$ if $\models_I b(\bar{s})$,

(iii) $\langle \text{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi } \Delta, \bar{s} \rangle \rightarrow \langle S_2 \textbf{ fi } \Delta, \bar{s} \rangle$ if $\not\models_I b(\bar{s})$,

(iv) $\langle \textbf{while } b \textbf{ do } S \textbf{ od } \Delta, \bar{s} \rangle \rightarrow \langle S; \textbf{while } b \textbf{ do } S \textbf{ od } \Delta, \bar{s} \rangle$ if $\models_I b(\bar{s})$,

(v) $\langle \textbf{while } b \textbf{ do } S \textbf{ od } \Delta, \bar{s} \rangle \rightarrow \langle \Delta, \bar{s} \rangle$ if $\not\models_I b(\bar{s})$,

(vi) $\langle fi\, \Delta, \bar{s} \rangle \rightarrow \langle \Delta, \bar{s} \rangle$,

(vii) $\langle ; \Delta, \bar{s} \rangle \rightarrow \langle \Delta, \bar{s} \rangle$.

Let $\rightarrow^*$ denote the transitive closure of $\rightarrow$. 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 $i = 0, 1, \ldots$, $s_i \rightarrow s_{i+1}$ holds. For simplicity we identify each finite sequence with its extension to the infinite execution sequence obtained by repeating the last state. Execution sequences are denoted by the letters σ, τ. If $\sigma = s_0, s_1, \ldots$, then by definition $\sigma[i] = s_i$.

For a program T we denote by Σ_T the set of all of its execution sequences. 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, $\sigma \in \Sigma_T$, and $i \geqslant 0$. We define

$\models_{T,I}$ at $S(\sigma, i)$ iff $\sigma[i] = \langle S; \Delta, \overline{\sigma[i]} \rangle$ for some Δ;

$\models_{T,I}$ after $S(\sigma, i)$ iff $\exists j < i$ $\langle S\Delta, \overline{\sigma[j]} \rangle = \sigma[j] \rightarrow^* \sigma[i] = \langle \Delta, \overline{\sigma[i]} \rangle$

for some Δ and if $\Delta \not\equiv E$ then for no k such that $j < k < i$ $\sigma[k] = \langle \Delta, \overline{\sigma[k]} \rangle$.

Intuitively $\models_{T,I}$ at $S(\sigma, i)$ holds if S is just to be executed in the state $\sigma[i]$ and $\models_{T,I}$ after $S(\sigma, i)$ holds if the execution of S has just terminated in $\sigma[i]$. While the definition of truth of at S is intuitively clear, that of after S requires some explanation.

The first line of the definition says the following. Some time *in the past* (in state $\sigma[j]$) the program S was just to be executed. Moreover, the remainder (Δ) which was still to be executed *after* S in $\sigma[j]$ is the program which now remains to be executed in $\sigma[i]$. This is a satisfactory definition if the remainder Δ does not begin with a **while** loop.

If Δ does begin with a **while** loop then we additionally claim that the execution of Δ did not start before $\sigma[i]$. Note that this condition is guaranteed if Δ does not begin with a **while** loop (provided $\Delta \not\equiv E$). This explains the second line of the definition.

Observe that by the definition $\models_{T,I}$ at $T(\sigma, i)$ iff $\sigma[i] = \langle T, \overline{\sigma[i]} \rangle$ and $\models_{T,I}$ after $T(\sigma, i)$ iff $\sigma[i] = \langle E, \overline{\sigma[i]} \rangle$.

Note that the definition of truth of at S depends only on the *current* state whereas the definition of truth of after S depends also on the *previous* states. This fact was not properly taken care of in the definition of semantics given in Apt and Delporte (1983).

We also introduce a formula $\text{at}^+ S$ which attempts to describe the fact that the control is at the beginning of S for the first time in the current round. If S does not begin with a **while** loop then by definition $\text{at}^+ S \equiv \text{at } S$. Otherwise $\text{at}^+ S \equiv \text{at}^+ S'$, where S begins by the **while** loop S'.

The form of $\text{at}^+ S$ for the **while** loop S depends on its direct context within T. It is defined as follows:

If S appears in T in the context:

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

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

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

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

If none of the above cases arise then T begins with S and we put $\text{at}^+ S \equiv \text{at } T$.

Introduction of the formula $\text{at}^+ S$ will allow us a more compact presentation of various proof rules and axioms.

The following example clarifies the introduced semantics of control formulas:

EXAMPLE. Let I be a standard interpretation in natural numbers and let $T \equiv \alpha: x = 1;$ **while** $x < 2$ **do**

$$S \equiv \textbf{if } x = 1 \textbf{ then } \beta: x := 0 \textbf{ else } \gamma: x := 2 \textbf{ fi}$$
$$\textbf{od}$$

Consider now some $\sigma \in \Sigma_T$. Then according to the definition

$\models_{T,I}$ at $x := 1\ (\sigma, 0)$,
$\models_{T,I}$ after $x := 1\ (\sigma, 1)$,
$\models_{T,I}$ at **while** $\cdots\ (\sigma, 2)$,
$\models_{T,I}$ at$^+$ **while** $\cdots\ (\sigma, 1)$,
$\models_{T,I}$ at $S\ (\sigma, 3)$,
$\models_{T,I}$ at $x := 0 (\sigma, 4)$,
$\models_{T,I}$ after $x := 0 (\sigma, 5)$,
$\models_{T,I}$ after $S (\sigma, 6)$,
$\not\models_{T,I}$ after $x := 2 (\sigma, 5)$ (since for no $j < 5$ $\models_{T,I}$ at $x := 2 (\sigma, j)$),
$\models_{T,I}$ at **while** $\cdots\ (\sigma, 7)$,
$\not\models_{T,I}$ after $x := 1 (\sigma, 6)$
(since $\models_{T,I}$ at **while** $\cdots\ (\sigma, 2)$ and $j = 0 < k = 2 < i = 6$),
$\not\models_{T,I}$at$^+$ **while** $\cdots\ (\sigma, 6)$
$\models_{T,I}$ at $S (\sigma, 8)$,
$\models_{T,I}$ at $x := 2 (\sigma, 9)$,
$\models_{T,I}$ after $x := 2 (\sigma, 10)$,
$\models_{T,I}$ after $S (\sigma, 11)$
$\not\models_{T,I}$ after $x := 0 (\sigma, 10)$
(since $\models_{T,I}$ at **while** $\cdots\ (\sigma, 7)$ and $j = 4 < k = 7 < i = 10$),
$\models_{T,I}$ at **while** $\cdots\ (\sigma, 12)$,
$\not\models_{T,I}$ after $x := 1 (\sigma, 11)$
(since $\models_{T,I}$ at **while** $\cdots\ (\sigma, 2)$ and $j = 0 < k = 2 < i = 1$),
$\models_{T,I}$ after **while** $\cdots\ (\sigma, 13)$,
$\models_{T,I}$ after $T (\sigma, 13)$.

Where for brevity we identified assignments with their labels. We hope that this example convinced the reader that the definitions of truth of at S and after S are natural. Our definition of truth of after S notably differs from that given in Owicki and Lamport (1982), which intuitively states that after S is true if the control within T is just after the program S. We shall disscuss the reasons for which we did not choose that definition after completing the definitions of semantics.

The above example also illustrates the fact that according to our definition of semantics different control points in the course of execution cannot coincide. We could have chosen another standpoint according to which various control points coincide. This would lead to more elegant semantics but to much more lengthly and complicated proofs. The above semantics is closer to the implementation level so it is more realistic. Moreover, the above choice does not result in a loss of any interesting properties.

We now continue with the definition of semantics. The truth of asser-

tions does not depend on the programs and we naturally put for $\sigma \in \Sigma_T$ and a formula p of L

$$\models_{T,I} p(\sigma, i) \quad \text{iff} \quad \models_I p(\overline{\sigma[i]}).$$

The truth of mixed formulas and other logical combinations of the control formulas as assertions is defined in a natural way.

Finally we define the truth of liveness formulas by putting for $\sigma \in \Sigma_T$

$$\models_{T,I} \mu_1 \rightsquigarrow \mu_2(\sigma, i) \quad \text{iff}$$
$$\models_{T,I} \mu_1(\sigma, i) \Rightarrow \exists j \geqslant i \models_{T,I} \mu_2(\sigma, j).$$

We now say that a formula φ *is true w.r.t. T and I,* written as $\models_{T,I} \varphi$ if for all $\sigma \in \Sigma_T$ and $i \geqslant 0 \models_{T,I} \varphi(\sigma, i)$ holds.

Having presented the definition of truth we now prove two useful lemmas concerning formulas of the form $\mathscr{C} \supset p$ and liveness formulas.

LEMMA 3.1. *Let S be a subprogram of T. Then for all formulas* $\mathscr{C} \supset p$ *and interpretations I*

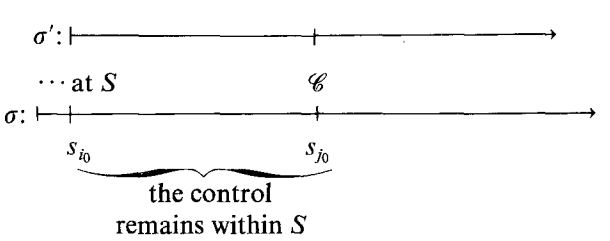
$$\models_{S,I} \mathscr{C} \supset p \quad \textit{implies} \quad \models_{T,I} \mathscr{C} \supset p.$$

Proof. We present only an informal argument leaving the formal details to the reader. $\mathscr{C}$ has to refer to a subprogram of S since otherwise $\models_{S,I} \mathscr{C} \supset p$ is not defined.

Given a computation σ of T we say that at the moment j the control in σ (or simply the control in (σ, j)) *resides within* S if $\models_{T,I} \mathscr{C}'(\sigma, j)$ holds for some $\mathscr{C}'$ refering to a subprogram of S.

Let $\sigma \in \Sigma_T$ and suppose that for some $j_0 \models_{T,I} \mathscr{C}(\sigma, j_0)$. Then at the moment j_0 the control in σ resides within S. Due to our definition of truth of control formulas there exists $i \leqslant j_0$ such that $\models_{T,I}$ at $S(\sigma, i)$ and for all j such that $i \leqslant j < j_0 \not\models_{T,I}$ after $S(\sigma, j)$. (Note that this would not be the case if we admitted **go-to** programs or other constructs breaking the control flow). Let i_0 be the least such i.

Consider now the computation σ' of S which starts in the assignation $\overline{\sigma[i_0]}$. The following diagram clarifies the definition of σ'.



The computation σ' passes in its first $j_0 - i_0$ steps through the same assignations and control points of S as σ starting from $\sigma[i_0]$. More formally, for every control formula $\mathscr{C}'$ refering to a subprogram of S $\overline{\sigma'[j]} = \overline{\sigma[i_0 + j]}$ and $\models_{S,I} \mathscr{C}'(\sigma', j)$ iff $\models_{T,I} \mathscr{C}'(\sigma, i_0 + j)$ for all $j \leqslant j_0 - i_0$.

In particular $\models_{S,I} \mathscr{C}(\sigma', j_0 - i_0)$. By the assumption $\models_{S,I} p(\sigma', j_0 - i_0)$, i.e., $\models_I p(\overline{\sigma'[j_0 - i_0]})$, so $\models_I p(\overline{\sigma[j_0]})$ and $\models_{T,I} p(\sigma, j_0)$. (Note that a crucial observation used in this proof was that in the computation σ the control resides within S for all moments j such that $i_0 \leqslant j \leqslant j_0$.) ∎

LEMMA 3.2. *Let S be a subprogram of T. Then for all liveness formulas φ and interpretations I*

$$\models_{S,I} \varphi \quad \textit{implies} \quad \models_{T,I} \varphi.$$

Proof. Once again we present only an informal proof. φ is of the form $\mathscr{C} \wedge p \rightsquigarrow \mathscr{C}' \wedge q$ for some control formulas $\mathscr{C}$ and $\mathscr{C}'$ referring to subprograms of S and some assertions p and q.

Suppose now that for some $\sigma \in \Sigma_T$ and i_0 $\models_{T,I} \mathscr{C} \wedge p(\sigma, i_0)$. We are to prove that for some $j_0 \geqslant i_0 \models_{T,I} \mathscr{C}' \wedge q(\sigma, j_0)$.

Similarly as in the proof of Lemma 3.1 there exists a computation σ' of S and i such that $\models_{S,I} \mathscr{C}(\sigma', i)$ and $\overline{\sigma'[i]} = \overline{\sigma[i_0]}$. Thus also $\models_{S,I} p(\sigma', i)$ since $\models_{T,I} p(\sigma, i_0)$, i.e., $\models_I p(\overline{\sigma[i_0]})$ holds. By the assumption for some $j \geqslant i \models_{S,I} \mathscr{C}' \wedge q(\sigma', j)$.

The computations σ and σ' pass through the same assignations and control points of S during the moments $l \in \{i_1, \ldots, i_0 + j - i\}$ and $l \in \{i, \ldots, j\}$, respectively. More formally for every control formula $\mathscr{C}''$ referring to a subprogram of S $\overline{\sigma[l]} = \overline{\sigma'[l + i - i_0]}$ and $\models_{T,I} \mathscr{C}''(\sigma, l)$ iff $\models_{S,I} \mathscr{C}''(\sigma', l + i - i_0)$ for all $l \in \{i_1, \ldots, i_0 + j - i\}$.

Thus $\models_{T,I} \mathscr{C}' \wedge q(\sigma, i_0 + j - i)$ because $\models_{S,I} \mathscr{C}' \wedge q(\sigma', j)$ and $\overline{\sigma[i_0 + j - i]} = \overline{\sigma'[j]}$. This concludes the proof. ∎

We now return to the discussion of the definition of truth of after S. Consider the programs $S \equiv \alpha\colon x := 1$ and $T \equiv S; S_1$, where

$$S_1 \equiv \textbf{while } x < 2 \textbf{ do}\beta\colon x := x + 1 \textbf{ od}.$$

Let I_N be the standard interpretation in natural numbers. Then $\models_{S,I_N}$ after $S \supset x = 1$ and $\models_{T,I_N}$ after $S \supset x = 1$.

However, according to the definition of truth of Owicki and Lamport (1982), after $S \supset x = 1$ holds in the context of the program S whereas it does not hold in the context of the program T. This follows from the fact that after $S \equiv$ at S_1 in the context of T. Thus Lemma 3.1 does not hold when the definition of truth of Owicki and Lamport (1982), is adopted. Also, in contrast to Owicki and Lamport (1982) most of the control points cannot coincide during the execution of a program.

Similarly Lemma 3.2 does not hold with the definitions of Owicki and Lamport (1982). To see this take $\varphi \equiv \text{after } S \wedge x > 0 \rightsquigarrow \text{after } S \wedge x = 1$. Then $\models_{S,I_N} \varphi$ and $\models_{T,I_N} \varphi$. Applying the definitions of Owicki and Lamport (1982) we get that φ holds in the context of S but it does not in the context of T since $\not\models_{T,I}$ at $S_1 \wedge x > 0 \rightsquigarrow$ at $S_1 \wedge x = 1$. The formulas of the form $\mathscr{C} \supset p$ will turn out to be needed in the proofs of liveness formulas. Lemma 3.1 and a proof theoretic counterpart of Lemma 3.2 will be needed in the completeness proof given later. The importance of Lemmas 3.1 and 3.2 lies in the fact that they allow to build proofs of desired formulas incrementally by a gradual extension of the context. This could not be done if the definitions of Owicki and Lamport (1982) were adopted. Note that according to our definitions the formula after $\alpha: x := 1 \supset x = 1$ is true *independently* of the context which is natural and desirable.

It should be noted that in our formalism more properties can be expressed than in that of Owicki and Lamport (1982), as their definition of "after S" can be easily expressed in our framework. We believe that our definitions of after S even though more complicated is more natural.

4. Expressiveness of the Syntax

A natural question to ask at this moment is what type of properties can be expressed using the syntax we introduced.

First of all it is perhaps useful to see why formulas of the form $\vdash_T \mathscr{C} \supset p$, $\vdash_T \mathscr{C} \wedge p \supset \mathscr{C}'$ and $\vdash_T \mathscr{C} \wedge p \supset \mathscr{C}' \wedge q$ are needed when studying liveness properties. Suppose that for some program T and interpretation I $\models_{T,I} \mathscr{C} \wedge p \supset \mathscr{C}' \wedge q$ which is the special case of $\models_{T,I} \mathscr{C} \wedge p \rightsquigarrow \mathscr{C}' \wedge q$. Then both $\models_{T,I} \mathscr{C} \wedge p \supset \mathscr{C}'$ and $\models_{T,I} \mathscr{C} \supset (p \supset q)$ hold. Now, a natural way to prove $\models_{T,I} \mathscr{C} \wedge p \rightsquigarrow \mathscr{C}' \wedge q$ is to establish first $\models_{T,I} \mathscr{C} \wedge p \supset \mathscr{C}'$ and $\models_{T,I} \mathscr{C} \supset (p \supset q)$, i.e., $\models_{T,I} \mathscr{C} \wedge p \supset q$, conclude from this $\models_{T,I} \mathscr{C} \wedge p \supset \mathscr{C}' \wedge q$ and finally derive from the last formula

$$\models_{T,I} \mathscr{C} \wedge p \rightsquigarrow \mathscr{C}' \wedge q.$$

Thus the formulas of the form $\vdash_T \mathscr{C} \supset p$, $\vdash_T \mathscr{C} \wedge p \supset \mathscr{C}'$ and $\vdash_T \mathscr{C} \wedge p \supset \mathscr{C}' \wedge q$ naturally arise in the proofs of liveness properties.

The formulas of the form $\vdash_T \mathscr{C} \supset q$ allow to express some restricted but nevertheless useful invariance properties.

Let I be a standard interpretation in natural numbers. Consider a program T executed over I.

Suppose that an assignment $\alpha: x := y/z$ is a subprogram of T and that we

wish to express the fact that whenever this assignment is executed y is divisible by z (z div y). This can be simply expressed by

$$\models_{T,I} \text{at } \alpha\colon x := y/z \supset (z \text{ div } y).$$

The statement that a variable x always remains within the bounds 1, max can be expressed by the conjunction of $\models_{T,I}$ after $S \supset 1 \leqslant x \leqslant \max$, where S ranges over assignments within T of the form $x := t$ or much more succintly by $\models_{T,I} 1 \leqslant x \leqslant \max$. (Note that $\models_{T,I} p$ is equivalent to the conjunction of $\models_{T,I} \mathscr{C} \supset p$ for $\mathscr{C}$ ranging over all control formulas referring to subprograms of T.)

Suppose now that we wish to express the fact that every variable of T gets initialized. To this purpose it is useful to introduce a special constant, say ω and extend the interpretation I to I_ω which has one new element $\boldsymbol{\omega}$ being the interpretation of ω and such that all terms containing variables which get value $\boldsymbol{\omega}$ in an assignation $\bar{s}$ evaluate in $\bar{s}$ to $\boldsymbol{\omega}$. Thus for example if $\bar{s}(x) = \boldsymbol{\omega}$ then $\bar{s}(x+1) = \boldsymbol{\omega}$. Suppose also that all execution sequences of T over I_ω start in an assignation in which all variables of T get value $\boldsymbol{\omega}$.

Then the property that all variables of T get initialized can be expressed as the conjunction of $\models_{T,I_\omega}$ at $\beta\colon x := t \supset t \neq \omega$ for all assignments $\beta\colon x := t$ being subprograms of T and $\models_{T,I_\omega}$ at $S \supset x_1 \neq \omega \wedge \cdots \wedge x_n \neq \omega$ for all subprograms S of T being of the form **if** b **then** S_1 **else** S_2 **fi** or **while** b **do** S_0 **od** where $x_1, \ldots, x_k$ is the list of all variables occurring in b.

Consider now liveness formulas. First, they can be used to express total correctness of programs—$\models_{T,I}$ at $T \wedge p \rightsquigarrow$ after $T \wedge q$ states that T is totally correct w.r.t. the assertions p and q.

Various other reachability statements can also be expressed. Consider for example a **while** loop S being a subprogram of T. Then the statement that S will eventually be exited whenever T is activated in an initial assignation satisfying p can be simply expressed as $\models_{T,I}$ at $T \wedge p \rightsquigarrow$ after S. More generally, reachability of a statement S under the initial assumption p is expressed by $\models_{T,I}$ at $T \wedge p \rightsquigarrow$ at S. Such reachability statements are especially useful when studying concurrent programs (see, e.g., Owicki and Lamport, 1982).

We note that liveness formulas, as already observed by Pnueli (1977), formalize the basic construct "if sometime p at l_0 then sometime (later) q at l_1" of the intermittent assertion method of Burstall (1974), (see also Manna and Waldinger, 1978). Here l_0, l_1 are labels attached to subprogams of the program in question. These constructs are especially useful when proving total correctness of a certain type of programs and correctness of continuously operating (or cyclic) programs.

It is interesting to note that partial correctness of programs cannot be expressed using the syntax of the paper. It turns out the simpler safety properties are already sufficient for proofs of liveness formulas.

5. A Proof System $\mathscr{S}$ For Proving Formulas of the Form $\vdash_T \mathscr{C} \supset p$

As a first step towards obtaining a proof system allowing to prove arbitrary liveness formulas we provide a proof system which deals with simple safety formulas of the form $\vdash_T \mathscr{C} \supset p$. We call this proof system $\mathscr{S}$.

Four types of formulas are allowed in the proof system: $\vdash_T \mathscr{C} \supset p$, $\vdash_T \mathscr{C} \wedge p \supset q$, $\vdash_T \mathscr{C} \supset \mathscr{C}'$ and p.

The system $\mathscr{S}$ consists of the following axioms and proof rules:

Assignment Rule. *Let $S \equiv \alpha: x := t$ be a subprogram of T,*

$$\text{R 1.} \quad \frac{\vdash_T \text{at } S \supset p[t/x]}{\vdash_T \text{after } S \supset p}.$$

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

Concatenation Axioms and Rule. *Let $S \equiv S_1; S_2$ be a subprogram of T,*

A 1. $\vdash_T \text{at } S \supset \text{at } S_1$,

A 2. $\vdash_T \text{at } S_1 \supset \text{at } S$,

A 3. $\vdash_T \text{after } S_2 \supset \text{after } S$,

A 4. $\vdash_T \text{after } S \supset \text{after } S_2$,

$$\text{R 2.} \quad \frac{\vdash_T \text{after } S_1 \supset p}{\vdash_T \text{at}^+ S_2 \supset p}.$$

Selection Rules. *Let $S \equiv$ **if** b **then** S_1 **else** S_2 **fi** be a subprogram of T,*

$$\text{R 3.} \quad \frac{\vdash_T \text{at } S \supset p}{\vdash_T \text{at}^+ S_1 \supset p \wedge b},$$

$$\text{R 4.} \quad \frac{\vdash_T \text{at } S \supset p}{\vdash_T \text{at}^+ S_2 \supset p \wedge \neg b},$$

$$\text{R 5.} \quad \frac{\vdash_T \text{after } S_1 \supset p, \vdash_T \text{after } S_2 \supset r}{\vdash_T \text{after } S \supset p \vee r}.$$

While Rules. *Let $S \equiv$ **while** b **do** S_0 **od** be a subprogram of T*

$$\text{R 6.} \quad \frac{\vdash_T \text{at } S \supset p}{\vdash_T \text{at}^+ S_0 \supset p \wedge b}$$

$$\text{R 7.} \quad \frac{\vdash_T \text{at } S \supset p}{\vdash_T \text{after } S \supset p \wedge \neg b}$$

$$\text{R 8.} \quad \frac{(\vdash_T \text{at}^+ S \supset p, \text{at } S \supset p \vdash_T \text{after } S_0 \supset p}{\vdash_T \text{at } S \supset p}.$$

The second premise of rule R 8 means that there exists a proof of $\vdash_T$ after $S_0 \supset p$ in the system $\mathscr{S}$ from the assumption $\vdash_T$ at $S \supset p$. This ensures that semantically p is an invariant of the body of the loop and expresses in the system a property corresponding to $\{p \wedge b\}\, S_0 \{p\}$ in the sense of Hoare's logic. Note however that for any I: $\vdash_T \{p \wedge b\}\, S_0 \{p\}$ implies $[\models_{T,I}$ at $S \supset p \Rightarrow \models_{T,I}$ after $S_0 \supset p]$ but not necessarily conversely.

Intuitively, the premises of rule R 8 state the following two facts. First, p holds when the control is for the first time at the beginning of the loop. Secondly, p remains invariant by each execution of the body loop.

Finally we have

INITIALIZATION AXIOM.

A 5. $\vdash_T$ at $T \supset$ **true**.

We also adopt without mentioning all axioms and proof rules of the classical logic concerning $\supset$ and $\wedge$ applied to formulas allowed in the system.

Recall that by Th(I) we denote the set of all assertions true in the interpretation I. We write $\vdash_{T,I} \mathscr{C} \supset p$ to denote the fact that there exists a proof of $\vdash_T \mathscr{C} \supset p$ in the system $\mathscr{S}$ which uses some assertions from Th(I) as axioms. If such a proof uses in addition the formula $\vdash_T \mathscr{C}' \supset q$ as an axiom then we write $\mathscr{C}' \supset q \vdash_{T,I} \mathscr{C} \supset p$.

6. SOUNDNESS AND ARITHMETICAL COMPLETENESS OF $\mathscr{S}$

The proof system $\mathscr{S}$ is sound in the sense of the following theorem:

THEOREM 1. *For any interpretation I, program T from $\mathscr{W}$ and a formula $\mathscr{C} \supset p$ if $\vdash_{T,I} \mathscr{C} \supset p$ then $\models_{T,I} \mathscr{C} \supset p$.*

A precise proof of this theorem requires use of the techniques similar to those of Section 3.7 of Apt (1981), to deal properly with rule R 8. The details are straightforward and omitted.

We now prove completeness of the system $\mathscr{S}$. Ideally, we would like to have an inverse of the implication stated in Theorem 1: $\models_{T,I} \mathscr{C} \supset p$ implies $\vdash_{T,I} \mathscr{C} \supset p$ for all interpretations I, programs T from $\mathscr{W}$ and formulas $\mathscr{C} \supset p$.

However, the proof requires definability of some assertions within I. Consequently the above implication can hold only for sufficiently "rich" interpretations. The situation is similar to the case of Hoare's logic explained for example in Section 2.7 of Apt (1981).

First we introduce the following definition.

DEFINITION. Let T be a program from $\mathscr{W}$, I an interpretation and $\mathscr{C}$ a

control formula. By the **filter** of $\mathscr{C}$ (w.r.t. T and I) we mean the following set of assignations

$$f_{T,I}(\mathscr{C}) = \{\bar{s} : \exists \sigma \in \Sigma_T \exists i (\models_{T,I} \mathscr{C}(\sigma, i) \wedge s = \sigma[i])\}$$

We also define the filter of a mixed formula $\mathscr{C} \wedge p$ by

$$f_{T,I}(\mathscr{C} \wedge p) = f_{T,I}(\mathscr{C}) \cap [p]_I,$$

where by definition $[p]_I = \{\bar{s} : \models_I p(\bar{s})\}$. Thus,

$$f_{T,I}(\mathscr{C} \wedge p) = \{\bar{s} : \exists \sigma \in \Sigma_T \exists i \models_{T,I} \mathscr{C} \wedge p(\sigma, i) \wedge s = \sigma[i])\}.$$

Now, the completeness proof we present relies on the definability of filters of all control formulas within I. This brings us to the notion of arithmetical interpretations defined in Harel (1979). We recall the definition:

Let L^+ be the minimal extension of the assertion language 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.

One of the examples of an arithmetical interpretation is of course the standard interpretation of L_p with nat (x) interpreted as $x = x$. Note that any interpretation of L with an infinite domain can be extended to an arithmetical interpretation of L^+.

We can now formulate the completeness theorem.

THEOREM 2. *Let I be an arithmetical interpretation and let T be a program from $\mathscr{W}$. Then for any formula $\mathscr{C} \supset p$, $\models_{T,I} \mathscr{C} \supset p$ implies $\vdash_{T,I} \mathscr{C} \supset p$.*

The proof of the theorem requires several lemmas. We start our analysis by investigating the notion of filters. It is easy to see that filters can be defined within arithmetical interpretations. Given an arithmetical interpretation I and a program T from $\mathscr{W}$ we denote by FIL$(\mathscr{C})$ a formula which defines the filter of $\mathscr{C}$ in I, i.e., such that $[\text{FIL}(\mathscr{C})]_I = f_{T,I}(\mathscr{C})$. Of course, FIL$(\mathscr{C})$ depends on T and I but we do not indicate this dependence to keep the notation simple. We also put FIL$(\mathscr{C} \wedge p) = \text{FIL}(\mathscr{C}) \wedge p$,

The following lemma summarizes the properties of the formulas FIL$(\mathscr{C})$.

LEMMA 6.1. *Let T be a program from $\mathscr{W}$ and I an arithmetical interpretation. Then we have*

(a) *for* $S \equiv \alpha\colon x := t$ *being a subprogram of* T

$$\models_{T,I} \mathrm{FIL}(\mathrm{at}\ S) \equiv \mathrm{FIL}(\mathrm{after}\ S)[t/x].$$

(b) *for* $S \equiv S_1; S_2$ *being a subprogram of* T

$$\models_{T,I} \mathrm{FIL}(\mathrm{at}\ S) \equiv \mathrm{FIL}(\mathrm{at}\ S_1),$$
$$\models_{T,I} \mathrm{FIL}(\mathrm{after}\ S_1) \supset \mathrm{FIL}(\mathrm{at}\ S_2),$$
$$\models_{T,I} \mathrm{FIL}(\mathrm{after}\ S_1) \equiv \mathrm{FIL}(\mathrm{at}^+\ S_2)$$
$$\models_{T,I} \mathrm{FIL}(\mathrm{after}\ S_2) \equiv \mathrm{FIL}(\mathrm{after}\ S).$$

(c) *for* $S \equiv$ **if** b **then** S_1 **else** S_2 **fi** *being a subprogram of* T

$$\models_{T,I} \mathrm{FIL}(\mathrm{at}\ S) \wedge b \supset \mathrm{FIL}(\mathrm{at}\ S_1),$$
$$\models_{T,I} \mathrm{FIL}(\mathrm{at}\ S) \wedge b \equiv \mathrm{FIL}(\mathrm{at}^+\ S_1)$$

analogous two conditions for S_2,

$$\models_{T,I} \mathrm{FIL}(\mathrm{after}\ S) \equiv \mathrm{FIL}(\mathrm{after}\ S_1) \vee \mathrm{FIL}(\mathrm{after}\ S_2).$$

(d) *for* $S \equiv$ **while** b **do** S_0 **od** *being a subprogram of* T

$$\models_{T,I} \mathrm{FIL}(\mathrm{at}\ S) \wedge b \supset \mathrm{FIL}(\mathrm{at}\ S_0),$$
$$\models_{T,I} \mathrm{FIL}(\mathrm{at}\ S) \wedge b \equiv \mathrm{FIL}(\mathrm{at}^+\ S_0)$$
$$\models_{T,I} \mathrm{FIL}(\mathrm{at}\ S) \equiv \mathrm{FIL}(\mathrm{at}^+\ S) \vee \mathrm{FIL}(\mathrm{after}\ S_0),$$
$$\models_{T,I} \mathrm{FIL}(\mathrm{after}\ S) \equiv \mathrm{FIL}(\mathrm{at}\ S) \wedge \neg b.$$

(e) $\models_{T,I} \mathrm{FIL}(\mathrm{at}\ T)$.

Proof. The proof is based on a straightforward analysis in terms of the filters $f_{T,I}(\mathscr{C})$ and is left to the reader. ∎

Next we introduce the following simple notion.

DEFINITION. A control formula $\mathscr{C}$ is **simple** if it is not of the form at S or after S for S being of the form $S_1; S_2$.

The relevance of the notion of filter is reflected by the following obvious lemma.

LEMMA 6.2. *Suppose that for all simple control formulas* $\mathscr{C}$ $\vdash_{T,I} \mathscr{C} \supset \mathrm{FIL}(\mathscr{C})$. *Then Theorem* 2 *holds*

Proof. First note that $\models_{T,I} \mathscr{C} \supset p$ implies $\models_I \mathrm{FIL}(\mathscr{C}) \supset p$. Thus if $\vdash_{T,I} \mathscr{C} \supset \mathrm{FIL}(\mathscr{C})$ then $\vdash_{T,I} \mathscr{C} \supset p$ by the classical logic.

Suppose now that $\models_{T,I} \mathscr{C} \supset p$ for some $\mathscr{C}$ which is not simple. Then for some simple $\mathscr{C}' \models_{T,I} \mathscr{C} \equiv \mathscr{C}'$ and $\models_{T,I} \mathscr{C}' \supset p$. By the above and the assumption $\vdash_{T,I} \mathscr{C}' \supset p$, so $\vdash_{T,I} \mathscr{C} \supset p$ by either Axiom A 1 or Axiom A 4. This concludes the proof. ∎

The above lemma leads us to the consideration of the formulas of the form $\mathscr{C} \supset \mathrm{FIL}(\mathscr{C})$ for simple $\mathscr{C}$.

We now introduce the following definition.

DEFINITION. We define a partial ordering $<$ on simple control formulas at S or after S where S is a subprogram of T as the least ordering such that:

(i) for S being an assignment statement

$$\text{at } S < \text{after } S,$$

(ii) for S being of the form $S_1; S_2$,

$$\text{after } S_1 < \text{at } S_2,$$

(iii) for S being of the form **if** b **then** S_1 **else** S_2 **fi**

$$\text{at } S < \text{at } S_1,$$

$$\text{at } S < \text{at } S_2,$$

$$\text{after } S_1 < \text{after } S,$$

$$\text{after } S_2 < \text{after } S,$$

(iv) for S being of the form **while** b **do** S_0 **od**

$$\text{at } S < \text{at } S_0,$$

$$\text{at } S < \text{after } S.$$

Note. The ordering $<$ corresponds to a natural ordering induced by the directed graph representing the flowchart of T with nodes being the control formulas and in which 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 are removed. The above clauses identify all pairs $\mathscr{C}, \mathscr{C}'$ such that $\mathscr{C}$ is a direct $<$ predecessor of $\mathscr{C}'$ which is the case when $\mathscr{C}'$ can be reached from $\mathscr{C}$ by executing T one step.

LEMMA 6.3. *$<$ is a well-ordering with the least element being* at S *where* $T \equiv S; S'$, S' *is possibly empty and* S *is not of the form* $S_1; S_2$.

Proof. Obvious. ∎

We now intend to prove the hypothesis of Lemma 6.2 by induction with respect to the well-ordering $<$ on simple control formulas. First we prove the following lemma.

LEMMA 6.4. *Suppose that $\mathscr{C}$ is a simple control formula which is not of the form at S for S being a* **while** *loop. Let $\{\mathscr{C}_i\}_{i=1,\ldots,m}$ be the set of all direct $<$ predecessors of $\mathscr{C}$. Then*

$$\{\mathscr{C}_i \supset \mathrm{FIL}(\mathscr{C}_i)\}_{i=1,\ldots,m} \vdash_{T,I} \mathscr{C} \supset \mathrm{FIL}(\mathscr{C}).$$

Proof. The proof is routine and we consider only a few selected cases. If $\mathscr{C}$ is of the form after S for S being of the form **if** b **then** S_1 **else** S_2 **fi** then $\mathscr{C}$ has exactly two direct predecessors: $\mathscr{C}_1 = \text{after } S_1$ and $\mathscr{C}_2 = \text{after } S_2$. Otherwise, it has exactly one predecessor $\mathscr{C}_1$ or zero if $\mathscr{C}$ is the $<$ least simple control formula.

Case I. $\mathscr{C}$ has no predecessors.

Then $\models_I \mathrm{FIL}(\mathscr{C}) \equiv \mathbf{true}$.

We have by Axiom A 2 $\vdash_{T,I} \mathscr{C} \supset \text{at } T$ so the claim follows by the initialization axiom and the above observation.

We now consider a case when $\mathscr{C}$ has exactly one predecessor.

Case II. For S being of the form **while** b **do** S_0 **od**, $\mathscr{C}_1 = \text{at } S$ and $\mathscr{C} = \text{at } S_0$. Since by assumption S_0 does not begin with a **while** loop, the claim follows by rule R 6 and the fact that $\models_I \mathrm{FIL}(\text{at } S_0) \equiv \mathrm{FIL}(\text{at } S) \wedge b$.

Finally we consider the case when $\mathscr{C}$ has two predecessors.

Case III. For S being of the form **if** b**then** S_1 **else** S_2 **fi**, $\mathscr{C}_1 = \text{after } S_1$, $\mathscr{C}_2 = \text{after } S_2$, and $\mathscr{C} = \text{after } S$.

The claim follows by rule R 5 and the fact that

$$\models_I \mathrm{FIL}(\text{after } S) \equiv \mathrm{FIL}(\text{after } S_1) \vee \mathrm{FIL}(\text{after } S_2).$$

The proofs of other cases are equally straightforward. ∎

The above lemma "almost" suffices to prove the hypothesis of Lemma 6.2. The only problem that remains is the case of simple control formulas of the form at S for S being a **while** loop.

To handle this case we need two more lemmas.

LEMMA 6.5. *Let $S \equiv$* **while** b **do** S_0 **od** *be a subprogram of T. Assume*

$$\text{at}^+ S_0 \supset \mathrm{FIL}(\text{at}^+ S_0) \vdash_{T,I} \text{after } S_0 \supset \mathrm{FIL}(\text{after } S_0).$$

Then

$$\text{at}^+ S \supset \mathrm{FIL}(\text{at}^+ S) \vdash_{T,I} \text{at } S \supset \mathrm{FIL}(\text{at } S).$$

Proof. We have $\text{at}^+ S_0 \equiv \text{at } S \wedge b$ so

$$\text{at } S \supset \text{FIL}(\text{at } S) \vdash_{T,I} \text{at}^+ S_0 \supset \text{FIL}(\text{at}^+ S_0).$$

Hence by the assumption

$$\text{at } S \supset \text{FIL}(\text{at } S) \vdash \text{after } S_0 \supset \text{FIL}(\text{after } S_0).$$

But $\models_I \text{FIL}(\text{after } S_0) \supset \text{FIL}(\text{at } S)$ so

$$\text{at } S \supset \text{FIL}(\text{at } S) \vdash_{T,I} \text{after } S_0 \supset \text{FIL}(\text{at } S).$$

Now by rule R 8 and the fact that $\models_I \text{FIL}(\text{at}^+ S) \supset \text{FIL}(\text{at } S)$ the claim follows. ∎

The next lemma proves the assumption of Lemma 6.5.

LEMMA 6.6. *For all subprograms S of T*

$$\text{at}^+ S \supset \text{FIL}(\text{at}^+ S) \vdash_{T,I} \text{after } S \supset \text{FIL}(\text{after } S).$$

Proof. The proof proceeds by induction on the structure of S. It is completely routine and left to the reader. ∎

Finally we prove

LEMMA 6.7. *Let $S \equiv$ **while** b **do** S_0 **od** be a subprogram of T and let $\mathscr{C}$ be a direct $<$ predecessor of* at S. *Then*

$$\mathscr{C} \supset \text{FIL}(\mathscr{C}) \vdash_{T,I} \text{at } S \supset \text{FIL}(\text{at } S).$$

Proof. By Lemmas 6.5 and 6.6 we get

$$\text{at}^+ S \supset \text{FIL}(\text{at}^+ S) \vdash_{T,I} \text{at } S \supset \text{FIL}(\text{at } S).$$

It is now enough to show that

$$\mathscr{C} \supset \text{FIL}(\mathscr{C}) \vdash_{T,I} \text{at}^+ S \supset \text{FIL}(\text{at}^+ S). \tag{5}$$

This can be shown by distinguishing two cases.

Case I. $\text{at}^+ S \equiv \mathscr{C}$. This case arises when S is preceded within T by another subprogram.

Case II. For some b $\text{at}^+ S \equiv \mathscr{C} \wedge b$. This case arises when S occurs within T within the context **while** b **do** $S(; S_1)$ **od**, **if** b **then** $S(; S_1)$ **else** S_2 **fi** or **if** b **then** S_1 **else** $S(; S_2)$ **fi**.

In both cases the claim is obvious. ∎

This brings us to the end of proof of Theorem 2. Namely due to Lemmas 6.4 and 6.7 we get the assumption of Lemma 6.2 by applying the

induction w.r.t. the ordering $<$ on simple control formulas. (The case when $\mathscr{C} \equiv$ at S for a **while** loop S such that $\mathscr{C}$ has no predecessors is handled as Case I of Lemma 6.4). Now thanks to Lemma 6.2 theorem holds.

7. A Proof System $\mathscr{L}$ for Proving Liveness Formulas

Finally, we present a proof system which is designed to prove the liveness formulas, i.e., formulas of the form $\vdash_T \mu_1 \rightsquigarrow \mu_2$. We call this system $\mathscr{L}$. In the system $\mathscr{L}$ we allow formulas of the form $\vdash_T \mu_1 \supset \mu_2$ and $\vdash_T \mu_1 \rightsquigarrow \mu_2$.

The system $\mathscr{L}$ contains all axioms and proof rules of $\mathscr{S}$ and additionally the following new axioms and rules which can be naturally divided into two parts. The first part consists of

Assignment Axiom. A 6. $\vdash_T$ at $S \wedge p[t/x] \rightsquigarrow$ after $S \wedge p$, *where* $S \equiv \alpha$: $x := t$ *is a subprogram of* T.

Concatenation Axiom. *Let* S_1, S_2 *be a subprogram of* T

A 7. $\vdash_T$ after $S_1 \wedge p \rightsquigarrow$ at $S_2 \wedge p$.

Selection Axioms and Rules. *Let* $S \equiv$ **if** b **then** S_1 **else** S_2 **fi** *be a subprogram of* T:

A 8. $\vdash_T$ at $S \wedge p \wedge b \rightsquigarrow$ at $S_1 \wedge p$

A 9. $\vdash_T$ at $S \wedge p \wedge \neg b \rightsquigarrow$ at $S_2 \wedge p$

A 10. $\vdash_T$ after $S_1 \wedge p \rightsquigarrow$ after $S \wedge p$

A 11. $\vdash_T$ after $S_2 \wedge p \rightsquigarrow$ after $S \wedge p$

R 9. $$\frac{\vdash_T \text{after } S_1 \supset p}{\vdash_T \text{after } S_2 \wedge \neg p \rightsquigarrow \text{after } S \wedge \neg p}$$

R 10. $$\frac{\vdash_T \text{after } S_2 \supset p}{\vdash_T \text{after } S_1 \wedge \neg p \rightsquigarrow \text{after } S \wedge \neg p}.$$

While Axioms and Rule. *Let* $S \equiv$ **while** b **do** S_0 **od** *be a subprogram of* T.

A 12. $\vdash_T$ after $S_0 \wedge p \rightsquigarrow$ at $S \wedge p$

A 13. $\vdash_T$ at $S \wedge p \wedge b \rightsquigarrow$ at $S_0 \wedge p$

A 14. $\vdash_T$ at $S \wedge p \wedge \neg b \rightsquigarrow$ after $S \wedge p$

R 11. $$\frac{\vdash_T \text{at } S \wedge p(n+1) \supset b,\ \vdash_T \text{at } S_0 \wedge p(n+1) \rightsquigarrow \text{after } S_0 \wedge p(n)}{\vdash_T \text{at } S \wedge \exists n p(n) \rightsquigarrow \text{at } S \wedge p(0)}.$$

The above axioms specify how the control moves through the program. Rule 11 shows how to prove the liveness properties of the **while** loops. It is an obvious adaption of the rule given in Harel (1979), appropriate for proving termination of **while** loops.

The second part axiomatizes the "$\rightsquigarrow$" operator and shows how to manipulate the liveness formulas. It consists of the following rules:

R 12. Reflexivity rule,

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

R 13. Transivity rule,

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

R 14. Confluence rule,

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

As usual, we also adopt all axioms and classical logic concerning $\supset$ and $\wedge$ and applied to all formulas allowed in the system $\mathscr{L}$.

8. Arithmetical Soundness and Completeness of $\mathscr{L}$

In order to prove soundness of the proof system $\mathscr{L}$ we should interpret its formulas in a model. However, not all models are appropriate here. The reason is that the **while** rule R 12 refers to natural numbers. To ensure a correct interpretation of this rule we should restrict ourselves to models which contain natural numbers. An appropriate class of such models is the one corresponding to arithmetical interpretations. Note that the system $\mathscr{L}$ is appropriate only for the assertion languages of the form L^+, and an expression such as $p(n+1)$ is actually a shorthand for

$$p(n+1) \wedge \text{nat}\,(n+1).$$

The appropriate formulation of the soundness of $\mathscr{L}$ is thus given by the following theorem.

Theorem 3. *Let I be an arithmetical interpretation and let T be a program from $\mathscr{W}$. Then for any liveness formula φ $\vdash_{T,I}$ implies $\models_{T,I} \varphi$.*

Here $\vdash_{T,I} \varphi$ stands of course, for the fact that there exists a proof of φ within $\mathscr{L}$ which uses some assertions from Th(I) as axioms.

To prove this theorem it suffices in view of Theorem 3 to prove validity

of the newly introduced axioms and soundness of the new while rule R 12. The proofs are straightforward and left to the reader.

We now turn to the issue of completeness of $\mathscr{L}$. Let $\mathscr{L}_1$ denote a subsystem of $\mathscr{L}$ which contains all axioms and proof rules which do not deal with formulas of the form $\vdash_T \mathscr{C} \supset p$. More precisely $\mathscr{L}_1$ consists of the axioms A 1–A 4 and A 6–A 13 proof rules R 9–R 11 and the classical logic part of $\mathscr{L}$. Thus $\mathscr{L}_1$ and $\mathscr{S}$ have in common only the concatenation axioms (apart of the classical logic part). We stress the hierarchical structure of $\mathscr{L}$ by proving a relative completeness of $\mathscr{L}_1$ w.r.t. $\mathscr{S}$. Let

$$\mathscr{S}(T, I) = \{\vdash_T \mathscr{C} \supset p: \models_{T,I} \mathscr{C} \supset p\} \cup \mathrm{Th(I)}.$$

Now by $\mathscr{S}(T, I) \vdash_{\mathscr{L}_1} \varphi$ we denote the fact that there exists a proof of $\vdash_T \varphi$ in $\mathscr{L}_1$ which uses some of the elements of $\mathscr{S}(T, I)$ as axioms.

We now prove the following theorem.

THEOREM 4. *Let I be an arithmetical interpretation and let T be a program from $\mathscr{W}$. Then for an liveness formula φ $\models_{T,I} \varphi$ implies $\mathscr{S}(T, I) \vdash_{\mathscr{L}_1} \varphi$.*

This theorem together with Theorem 2 implies the following corollary.

COROLLARY (arithmetical completeness of $\mathscr{L}$). *Let I be an arithmetical interpretation and let T be an program from $\mathscr{W}$. Then for any liveness formula φ $\models_{T,I} \varphi$ implies $\vdash_{T,I} \varphi$.*

The proof of the theorem relies on the following important lemma which is a proof theoretic counterpart of Lemma 3.2.

LEMMA 8.1. *Let S be a subprogram of T and I an interpretation. Then for any liveness formula φ $\mathscr{S}(S, I) \vdash_{\mathscr{L}_1} \varphi$ implies $\mathscr{S}(T, I) \vdash_{\mathscr{L}_1} \varphi$.*

Proof. Consider a proof of $\vdash_S \varphi$ in $\mathscr{L}_1$ from the set of assumptions $\mathscr{S}(S, I)$. By Lemma 3.2 all these assumptions are also elements of $\mathscr{S}(T, I)$. Now replacing everywhere in the above proof "$\vdash_S$" by "$\vdash_T$" we obtain a proof of $\vdash_T \varphi$ in $\mathscr{L}_1$ from the set of assumptions $\mathscr{S}(T, I)$. ∎

We shall also need the following lemmas and notions.

LEMMA 8.2. *For any program T and arithmetical interpretation I $\models_{T,I} \mathscr{C} \wedge p \supset \mathscr{C}' \wedge q$ implies $\mathscr{S}(T, I) \vdash_{\mathscr{L}_1} \mathscr{C} \wedge p \rightsquigarrow \mathscr{C}' \wedge q$.*

Proof. It is easy to see that $\mathscr{S}(T, I) \vdash_{\mathscr{L}_1} \mathscr{C} \supset \mathscr{C}'$. Also

$$\mathscr{S}(T, I) \vdash_{\mathscr{L}_1} \mathscr{C} \wedge p \supset q.$$

Thus by the classical logic $\mathscr{S}(T, I) \vdash_{\mathscr{L}_1} \mathscr{C} \wedge p \supset \mathscr{C}' \wedge q$ so $\mathscr{S}(T, I) \vdash_{\mathscr{L}_1} \mathscr{C} \wedge p \rightsquigarrow \mathscr{C}' \wedge q$ by the reflexivity rule R 12 ∎

For a subprogram S of T let $\mathscr{C}(S)$ stand for the set of all control formulas refering to a subprogram of S.

LEMMA 8.3. *Let S be a subprogram of T and let I be an arithmetical interpretation. Suppose that for all liveness formulas φ $\models_{S,I} \varphi$ implies $\mathscr{S}(S, I) \vdash_{\mathscr{L}_1} \varphi$. Then for any $\mathscr{C}, \mathscr{C}' \in \mathscr{C}(S)$ and assertions p and q if*

$$\forall \sigma \in \Sigma_T \forall i[\models_{T,I} \mathscr{C} \wedge p(\sigma, i) \Rightarrow$$
$$\exists j \geqslant i[\models_{T,I} \mathscr{C} \wedge p(\sigma, j) \wedge \forall k(i \leqslant k < j \Rightarrow \not\models_{T,I} \text{after } S(\sigma, k)]]] \quad (1)$$

then $\mathscr{S}(T, I) \vdash_{\mathscr{L}_1} \mathscr{C} \wedge p \rightsquigarrow \mathscr{C}' \wedge q$.

Proof. Note that the condition (1) simply states that $\mathscr{C} \wedge p$ "leads to" $\mathscr{C}' \wedge q$ without leaving S.

Consider the following set of states:

$$A = \{s: \exists \sigma \in \Sigma_T \exists i, j[i < j \wedge \models_{T,I} \mathscr{C} \wedge p(\sigma, i)$$
$$\wedge \not\models_{T,I} \mathscr{C}' \wedge q(\sigma, i) \wedge \models_{T,I} \mathscr{C}' \wedge q(\sigma, j) \wedge s = \sigma[i]]\}.$$

Since I is arithmetical, there exists an assertion r which defines $\bar{A}$ in I. It is easy to see that

$$\models_{T,I} \mathscr{C} \wedge p \wedge \neg r \supset \mathscr{C}' \wedge q.$$

By Lemma 8.2

$$\mathscr{S}(T, I) \vdash_{\mathscr{L}_1} \mathscr{C} \wedge p \wedge \neg r \rightsquigarrow \mathscr{C}' \wedge q. \quad (2)$$

We now show that

$$\models_{S,I} \mathscr{C} \wedge p \wedge r \rightsquigarrow \mathscr{C}' \wedge q \quad (3)$$

It is instructive to note that $\models_{S,I} \mathscr{C} \wedge p \rightsquigarrow \mathscr{C}' \wedge q$ does not need to hold. To see this consider the program $T \equiv \alpha: x := 1;\ S$ where $S \equiv \beta: x := x + 1$. Then for the standard interpretation I_N

$$\models_{T,I_N} \text{at } S \wedge \textbf{true} \rightsquigarrow \text{after } S \wedge x = 2 \text{ but obviously}$$

$\not\models_{S,I_N}$ at $S \wedge$ **true** $\rightsquigarrow$ after $S \wedge x = 2$. Note that (3) holds for $\mathscr{C} \equiv$ at S, $\mathscr{C}' \equiv$ after S, $p \equiv$ **true** and $q \equiv x = 2$. We have here $r \equiv x = 1$ and

$$\models_{S,I_N} \text{at } S \wedge x = 1 \rightsquigarrow \text{after } S \wedge x = 2.$$

However there are more subtle cases which require to exercise some caution. Let $T \equiv \alpha: x := 1;\ S$ where $S \equiv$ **while** $x < 2$ **do** $\beta: x := x + 1$ **od**. Then $\models_{T,I_N}$ at $S \wedge x \leqslant 2 \rightsquigarrow$ after $\beta: x := x + 1$ whereas obviously

$$\not\models_{S,I_N} \text{at } S \wedge x \leqslant 2 \rightsquigarrow \text{after } \beta: x := x + 1$$

(take the initial state with $x = 2$). We have here $r \equiv x \leqslant 1$ and $\models_{S,I_N}$ at $S \wedge x \leqslant 1 \rightsquigarrow$ after $\beta: x := x + 1$.

To prove (3) take some $\sigma' \in \Sigma_S$ and suppose that for some $k \models_{S,I} \mathscr{C} \wedge p \wedge r(\sigma', k)$. Since $\models_I r(\overline{\sigma'[k]})$, by the definition of r there

exists $\sigma \in \Sigma_T$ and i such that $\overline{\sigma[i]} = \overline{\sigma'[k]}$, $\models_{T,I} \mathscr{C}(\sigma, i)$ and $\not\models_{T,I} \mathscr{C}' \wedge q(\sigma, i)$. Since $\models_I p \wedge r(\overline{\sigma'[k]})$, we actually have

$$\models_{T,I} \mathscr{C} \wedge p \wedge r(\sigma, i).$$

By (1) there exists j satisfying the appropriate conditions. The computations σ and σ' "pass through" the same assignations and control points of S during the moments $i+1,\ldots, j$ and $k+1,\ldots, j+k-i$, respectively.

Thus $\models_{S,I} \mathscr{C}' \wedge q(\sigma', j+k-i)$ since $\models_{T,I} \mathscr{C}' \wedge q(\sigma, j)$.

This proves (3). Now by the assumption of the lemma

$$\mathscr{S}(S, I) \vdash_{\mathscr{L}_1} \mathscr{C} \wedge p \wedge r \rightsquigarrow \mathscr{C}' \wedge q.$$

By Lemma 8.1

$$\mathscr{S}(T, I) \vdash_{\mathscr{L}_1} \mathscr{C} \wedge p \wedge r \rightsquigarrow \mathscr{C}' \wedge q \tag{4}$$

Now (3) and (4) imply the claim by the confluence rule R 15. ∎

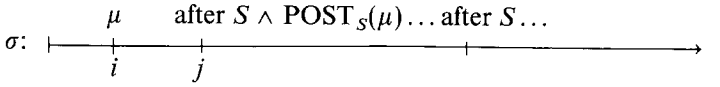
Finally we introduce the following notions.

DEFINITION. Let S be a subprogram of T, I an arithmetical interpretation and μ a mixed formula. We define the following two sets of states.

$$\begin{aligned}
\mathrm{pre}_S(\mu) = \{s : \forall \sigma \in \Sigma_T \forall i[(s = \sigma[i] \wedge \models_{T,I} \text{at } S(\sigma, i)) \\
\Rightarrow \exists j[i \leqslant j \wedge \models_{T,I} \mu(\sigma, j) \wedge \forall k(i < k \leqslant j \Rightarrow \not\models_{T,I} \text{at } S(\sigma, k))]]\}, \\
\mathrm{post}_S(\mu) = \{s : \exists \sigma \in \Sigma_T \exists i, j \\
[i \leqslant j \wedge \models_{T,I} \mu(\sigma, i) \wedge \models_{T,I} \text{after } S(\sigma, j) \wedge \forall k \\
i < k \leqslant j \Rightarrow \not\models_{T,I} \text{after } S(\sigma, k) \wedge s = \sigma[j]]\}
\end{aligned}$$

Of course both sets depend on T and I. Since I is arithmetical, there exists assertions $\mathrm{PRE}_S(\mu)$ and $\mathrm{POST}_S(\mu)$ which define in I $\overline{\mathrm{pre}}_S(\mu)$ and $\overline{\mathrm{post}}_S(\mu)$, respectively.

The following diagrams clarify the definitions of $\mathrm{PRE}_S(\mu)$ and $\mathrm{POST}_S(\mu)$:



The following lemma summarizes the properties of $\mathrm{PRE}_S(\mu)$ and $\mathrm{POST}_S(\mu)$.

LEMMA 8.4. *Let S, T, I and μ be as above. Then*

(a) $\models_{T,I} \text{at } S \wedge \text{PRE}_S(\mu) \rightsquigarrow \mu$,

(b) *If* $\models_{T,I} \mu \rightsquigarrow \text{after } S$ *then* $\models_{T,I} \mu \rightsquigarrow \text{after } S \wedge \text{POST}_S(\mu)$.

Proof. Left to the reader. ∎

We now proceed with the proof of Theorem 6. Suppose that $\models_{T,I} \mathscr{C} \wedge p \rightsquigarrow \mathscr{C}' \wedge q$. By Lemma 8.2 we can assume that

$$\not\models_{T,I} \mathscr{C} \wedge p \supset \mathscr{C}' \wedge q. \qquad (*)$$

We now prove the claim by induction on the structure of T. For each type of T we first list all the cases which have to be considered.

Case I. T is of the from $\alpha: x := t$.

$$\mathscr{C} \equiv \text{at } T, \qquad \mathscr{C}' \equiv \text{after } T.$$

Case II. T is of the form $S_1; S_2$

(a) $\mathscr{C} \equiv \text{at } T, \mathscr{C}' \equiv \text{at } T$,

(b) $\mathscr{C} \equiv \text{at } T, \mathscr{C}' \in \mathscr{C}(S_1)$,

(c) $\mathscr{C} \equiv \text{at } T, \mathscr{C}' \in \mathscr{C}(S_2)$,

(d) $\mathscr{C} \equiv \text{at } T, \mathscr{C}' \equiv \text{after } T$,

(e) $\mathscr{C}, \mathscr{C}' \in \mathscr{C}(S_1)$,

(f) $\mathscr{C}, \mathscr{C}' \in \mathscr{C}(S_2)$,

(g) $\mathscr{C} \in \mathscr{C}(S_1), \mathscr{C}' \equiv \text{after } T$,

(h) $\mathscr{C} \in \mathscr{C}(S_1), \mathscr{C}' \in \mathscr{C}(S_2)$,

(i) $\mathscr{C} \in \mathscr{C}(S_2), \mathscr{C}' \equiv \text{after } T$.

Case III. T is of the form **if** b **then** S_1 **else** S_2 **fi**. Let $i \in \{1, 2\}$.

(a) $\mathscr{C} \equiv \text{at } T, \mathscr{C}' \in \mathscr{C}(S_i)$,

(b) $\mathscr{C} \equiv \text{at } T, \mathscr{C}' \equiv \text{after } T$,

(c) $\mathscr{C}, \mathscr{C}' \in \mathscr{C}(S_i)$,

(d) $\mathscr{C} \in \mathscr{C}(S_i), \mathscr{C}' \equiv \text{after } T$.

Case IV. T is of the form **while** b **do** S **od**

(a) $\mathscr{C} \equiv \text{at } T, \mathscr{C}' \equiv \text{at } T$,

(b) $\mathscr{C} \equiv \text{at } T, \mathscr{C}' \equiv \text{after } T$,

(c) $\mathscr{C}, \mathscr{C}' \in \mathscr{C}(S)$,

(d) $\mathscr{C} \equiv \text{at } T, \mathscr{C}' \in \mathscr{C}(S)$,

(e) $\mathscr{C} \in \mathscr{C}(S), \mathscr{C}' \equiv \text{at } T$,

(f) $\mathscr{C} \in \mathscr{C}(S), \mathscr{C}' \equiv \text{after } T$.

The other cases cannot arise since we assumed (*). For example if in Case III $\mathscr{C} \in \mathscr{C}(S_i)$ and $\mathscr{C}' \in \mathscr{C}(S_{3-i})$ then necessarily $\models_{T,I} \mathscr{C} \wedge p \supset \textbf{false}$ so $\models_{T,I} \mathscr{C} \wedge p \supset \mathscr{C}' \wedge q$.

We now proceed with the proofs of selected cases. To simplify the notation we write $\vdash_T \varphi$ instead of $\mathscr{S}(T, I) \vdash_{\mathscr{L}_1} \varphi$.

Case II. First note that we can reduce the cases refering to at T or after T to the other ones. Indeed, we have $\models_{T,I}$ at $T \equiv$ at S_1,

$$\models_{T,I} \text{after } T \equiv \text{after } S_2$$

and also by the concatenation Axioms A 1–A 4 $\vdash_T$ at $T \equiv$ at S_1 and $\vdash_T$ after $T \equiv$ after S_2.

We are thus left with cases (e), (f), and (h):

ad (e) We then have $\models_{S_1,I} \mathscr{C} \wedge p \rightsquigarrow \mathscr{C}' \wedge q$ so by the induction hypothesis $\vdash_{S_1} \mathscr{C} \wedge p \rightsquigarrow \mathscr{C}' \wedge q$ and by Lemma 8.1 $\vdash_T \mathscr{C} \wedge p \rightsquigarrow \mathscr{C}' \wedge q$.

ad (f) Clearly (1) holds for $S \equiv S_2$. Also thanks to the induction hypothesis the other assumption of Lemma 8.3 holds. The claim is now the conclusion of Lemma 8.3.

ad (h) Any execution sequence $\sigma \in \Sigma_T$ has to "pass through" after S_1 and at S_2 in order to "realize" $\mathscr{C}' \wedge q$ after having "realized" $\mathscr{C} \wedge p$. We thus have $\models_{T,I} \mathscr{C} \wedge p \rightsquigarrow$ after S_1 and by Lemma 11.4

$$\models_{T,I} \mathscr{C} \wedge p \rightsquigarrow \text{after } S_1 \wedge \text{POST}_{S_1}(\mathscr{C} \wedge p).$$

We get now by Case II(e)

$$\vdash_T \mathscr{C} \wedge p \rightsquigarrow \text{after } S_1 \wedge \text{POST}_{S_1}(\mathscr{C} \wedge p). \tag{5}$$

Also it is easy to see that $\models_{T,I}$ at $S_2 \wedge \text{POST}_{S_2}(\mathscr{C} \wedge p) \rightsquigarrow \mathscr{C}' \wedge q$. We get now by Case II(f)

$$\vdash_T \text{at } S_2 \wedge \text{POST}_{S_1}(\mathscr{C} \wedge p) \rightsquigarrow \mathscr{C}' \wedge q. \tag{6}$$

Finally by Axiom A 7, classical logic, the reflexivity rule R 12 and the transitivity rule R 13

$$\vdash_T \text{after } S_1 \wedge \text{POST}_{S_1}(\mathscr{C} \wedge p) \rightsquigarrow \text{at } S_2 \wedge \text{POST}_{S_2}(\mathscr{C} \wedge p). \tag{7}$$

Combining (5)–(7) by the transitivity rule R 13 we get the desired result.

Case III. ad (a) Suppose $i = 1$. We clearly have

$$\models_{S_1,I} \text{at } S_1 \wedge p \wedge b \rightsquigarrow \mathscr{C}' \wedge q.$$

Thus by the induction hypothesis and Lemma 8.1

$$\vdash_T \text{at } S_1 \wedge p \wedge b \rightsquigarrow \mathscr{C}' \wedge q.$$

Also by the selection Axiom A 8 used with $p \wedge b$ in place of p $\vdash_T$ at $T \wedge p \wedge b \rightsquigarrow$ at $S_1 \wedge p \wedge b$. Thus by the transitivity rule R 13

$$\vdash_T \text{at } T \wedge p \wedge b \rightsquigarrow \mathscr{C}' \wedge q. \tag{8}$$

Also note that $\models_{T,I}$ at $T \wedge p \rightsquigarrow \mathscr{C}' \wedge q$ implies $\models_{T,I}$ at $T \wedge p \rightsquigarrow$ at S_1 so $\models_{T,I}$ at $T \wedge p \supset b$. Thus $\vdash_T$ at $T \wedge p \supset b$ so by the classical logic and the reflexivity rule R 12

$$\vdash_T \text{at } T \wedge p \rightsquigarrow \text{at } T \wedge p \wedge b. \tag{9}$$

Combining (8) and (9) by the transitivity rule R 13 we get now the desired result.

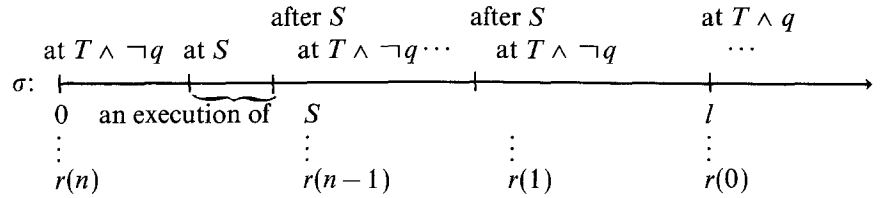
For $i = 2$ the reasoning is analogous.

Case IV ad (a) Let n be a fresh variable. Consider the following set of states: $C = \{s: \models_I \text{nat}(n)(\bar{s}) \wedge \exists \sigma \in \Sigma_T [s = \sigma[0] \wedge \models_{T,I} \text{at } T(\sigma, 0) \wedge \exists l [\models_{T,I} \text{at } T \wedge q(\sigma, l) \wedge \forall j < l \not\models_{T,I} \text{at } T \wedge q(\sigma, j) \wedge \#\{j: j < l \wedge \models_{T,I} \text{at } T(\sigma, j)\} = k]]$, where $k = \bar{s}(n)\}$.

By $\#K$ we denote the cardinality of the set K. Thus $s \in C$ iff $\bar{s}(n)$ is a natural number, say k, such that the loop in T is executed exactly k times when started in the assignation $\bar{s}$ before reaching for the first time the assignation $\bar{s}_1$ which satisfies q.

Since the interpretation I is arithmetical there exists an assertion $r(n)$ which defines $\bar{C}$ in I.

The following diagram illustrates the definition of $r(n)$:



One can easily check that

$$\models_{T,I} \text{at } T \wedge r(n+1) \supset b,$$

$$\models_{S,I} \text{at } S \wedge r(n+1) \rightsquigarrow \text{after } S \wedge r(n),$$

$$\models_{T,I} \text{at } T \wedge r(0) \supset q,$$

and

$$\models_{T,I} \text{at } T \wedge p \supset \exists n r(n).$$

We thus have $\vdash_T$ at $T \wedge r(n+1) \supset b$; by the induction hypothesis and Lemma 11.1 $\vdash_T$ at $S \wedge r(n+1) \rightsquigarrow$ after $S \wedge r(n)$ and $\vdash_T$ at $T \wedge r(0) \supset q$ and $\vdash_T$ at $T \wedge p \supset \exists n r(n)$. The first two formulas imply by the **while**-rule R 12 $\vdash_T$ at $T \wedge \exists n r(n) \rightsquigarrow$ at $T \wedge r(0)$. The other two formulas imply by

rule R 13 and the classical logic $\vdash_T \text{at } T \wedge r(0) \rightsquigarrow \text{at } T \wedge q$ and $\vdash_T \text{at } T \wedge p \rightsquigarrow \text{at } T \wedge \exists n r(n)$. Combining the last three formulas by the transitivity rule R 13 we get the desired result.

ad (c). This is the case with the most complicated proof. Consider the following set of states

$$A = \{s : \exists \sigma \in \Sigma_T \exists i, j, k [i \leqslant j < k \wedge \models_{T,I} \mathscr{C} \wedge p(\sigma, i) \wedge \models_{T,I} \text{after } S(\sigma, j) \wedge \models_{T,I} \mathscr{C}' \wedge q(\sigma, k) \wedge \forall l (i \leqslant l < k \Rightarrow \not\models_{T,I} \mathscr{C}' \wedge q(\sigma, l)) \wedge s = \sigma[i]]\}.$$

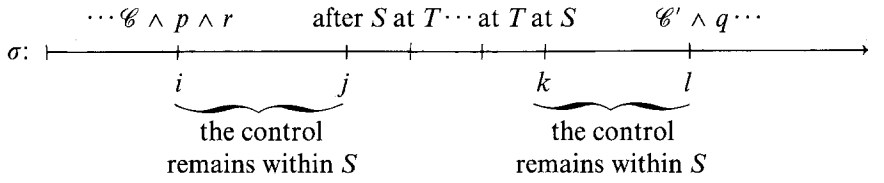
Thus $s \in A$ iff there exists an execution sequence σ which reaches s at a moment i and

(a) at this moment $\mathscr{C} \wedge p$ holds,

(b) at some later moment $\mathscr{C}' \wedge q$ holds, but only after the control has temporarily left S.

Because I is arithmetical, there exists an assertion r which defines $\bar{A}$ in I. It is easy to see by the definition of r that any execution sequence $\sigma \in \Sigma_T$ which "reaches" $\mathscr{C} \wedge p \wedge r$ will also "reach" $\mathscr{C}' \wedge q$ after the control has temporarily left S, i.e., after some number passes "through" at T.

Thus each execution sequence σ from Σ_T which reaches $\mathscr{C} \wedge p \wedge r$ is of the form



This implies by the definition of $\text{POST}_S(\mu)$ and $\text{PRE}_S(\mu)$ that

$$\models_{T,I} \mathscr{C} \wedge p \wedge r \rightsquigarrow \text{after } S \wedge \text{POST}_S(\mathscr{C} \wedge p \wedge r), \tag{10}$$

$$\models_{T,I} \text{after } S \wedge \text{POST}_S(\mathscr{C} \wedge p \wedge r) \rightsquigarrow \text{at } T \wedge \text{POST}_S(\mathscr{C} \wedge p \wedge r), \tag{11}$$

$$\models_{T,I} \text{at } T \wedge \text{POST}_S(\mathscr{C} \wedge p \wedge r) \rightsquigarrow \text{at } T \wedge \text{PRE}_S(\mathscr{C}' \wedge q), \tag{12}$$

$$\models_{T,I} \text{at } T \wedge \text{PRE}_S(\mathscr{C}' \wedge q) \rightsquigarrow \text{at } S \wedge \text{PRE}_S(\mathscr{C}' \wedge q), \tag{13}$$

$$\models_{T,I} \text{at } S \wedge \text{PRE}_S(\mathscr{C}' \wedge q) \rightsquigarrow \mathscr{C}' \wedge q. \tag{14}$$

We now show that everywhere in (10)–(14) we can replace $\models_{T,I}$ by $\vdash_T$.

Consider (10) first. It is easy to see that (1) is satisfied for the appropriate control formulas and assertions. We can now replace $\models_{T,I}$ by $\vdash_T$ thanks to the induction hypothesis and Lemma 8.3.

In (11) we can replace $\models_{T,I}$ by $\vdash_T$ due to the while Axiom A 12.

In (12) we can replace $\models_{T,I}$ by $\vdash_T$ by the Case IV(a).

To deal with (13) first note that (13) implies

$$\models_{T,I} \text{at } T \wedge \text{PRE}_S(\mathscr{C}' \wedge q) \supset b.$$

Thus by the classical logic and the reflexivity rule R 12

$$\vdash_T \text{at } T \wedge \text{PRE}_S(\mathscr{C}' \wedge q) \rightsquigarrow \text{at } T \wedge \text{PRE}_S(\mathscr{C}' \wedge q) \wedge b.$$

On the other hand by the while Axiom A 13,

$$\vdash_T \text{at } T \wedge \text{PRE}_S(\mathscr{C}' \wedge q) \wedge b \rightsquigarrow \text{at } S \wedge \text{PRE}_S(\mathscr{C}' \wedge q).$$

Thus by the transitivity rule R 13

$$\vdash_T \text{at } T \wedge \text{PRE}_S(\mathscr{C}' \wedge q) \rightsquigarrow \text{at } S \wedge \text{PRE}_S(\mathscr{C}' \wedge q) \text{ as desired.}$$

Finally, to deal with (14) it is sufficient to observe that (1) holds for the appropriate assertions and by the induction hypothesis the conclusion of Lemma 8.3 holds.

Combining now (10)–(14) with $\models_{T,I}$ replaced by $\vdash_T$ we get

$$\vdash_T \mathscr{C} \wedge p \wedge r \rightsquigarrow \mathscr{C}' \wedge q. \tag{15}$$

On the other hand, by the definition of r the condition (1) is satisfied with p replaced by $p \wedge \neg r$. We thus have by the induction hypothesis and Lemma 8.3,

$$\vdash_T \mathscr{C} \wedge p \wedge \neg r \rightsquigarrow \mathscr{C}' \wedge q. \tag{16}$$

The claim now follows from (15) and (16) by the confluence rule R 14.

The proofs of other cases are similar and omitted. This concludes the proof of Theorem 4. ∎

9. Conclusions

The axioms and proof rules we provided allow to formalize in a natural way informal proofs of (here considered) safety properties and liveness properties. Moreover the completeness proofs provide heuristics which can be helpful when trying to prove specific properties.

For certain type of properties some other axioms and proof rules are useful or needed and their use can lead to simpler and shorter proofs.

First, consider safety properties discussed in Section 4.

To prove statements of the form $\vdash_T p$, like for example $\vdash_T 1 \leqslant x \leqslant \max$, we need the following obviously sound rule

$$\frac{\vdash_T \mathscr{C} \supset p \text{ for all } \mathscr{C} \in \mathscr{C}(T)}{\vdash_T p}$$

(Recall that $\mathscr{C}(T)$ stands for the set of all control formulas referring to a subprogram of T).

To deal adequately with the statement that all variables $x_1,\ldots, x_k$ of T get initialized only interpretations of the form I_ω and execution sequences starting in assignations $\bar{s}$ such that $\bar{s}(X_1) = \cdots = \bar{s}(x_k) = \omega$ should be considered.

Consider now liveness properties. A closer look at the correctness proofs provided in Burstall (1974) and Manna and Waldinger (1978) shows that transfinite induction and an instantiation rule applied to liveness formulas is used there. These rules are absent in our repertoire. They have the following form:

INDUCTION RULE.

$$\frac{(\forall \beta < \alpha \vdash_T \varphi(\beta)) \supset \vdash_T \varphi(\alpha)}{\vdash_T \varphi(\alpha)}$$

where α, β range over ordinals (or a well-ordering) and do not occur in T.

INSTANTIATION RULE.

$$\frac{\vdash_T \varphi(x)}{\vdash_T \varphi(t)}$$

where x does not occur in T.

In both cases φ stands for a liveness formula.

A successful use of the intermittent assertion method seems to hinge on the use of the above two proof rules.

Once the induction rule is allowed a simpler completeness proof of the system $\mathscr{L}$ can be given following the lines of the proof given in Pnueli (1977). Such proofs however, are not syntax directed and consequently are only of a limited interest.

If we are interested in proving more complicated safety properties then first the syntax should be extended by allowing the "always" operator "$\Box \varphi$" defined by

$$\models_{T,I} \Box \varphi(\sigma, i) \quad \text{iff} \quad \forall j \geqslant i \models_{T,I} \varphi(\sigma, j).$$

Using this operator partial correctness of T w.r.t. p and q can be expressed by $\models_{T,I}$ at $T \wedge p \supset \Box$ (after $T \supset q$).

A proof system adequate to prove partial correctness can be obtained by a simple modification of the system $\mathscr{S}$. It suffices to replace every axiom $\vdash_T \varphi$ by $\vdash_T$ at $T \wedge p \supset \Box \varphi$ and all rules of the form

$$\frac{\vdash_T \varphi_1,\ldots,\vdash_T \varphi_k}{\vdash_T \varphi}$$

by

$$\frac{\vdash_T \text{at } T \wedge p \supset \Box \varphi_1, \ldots, \vdash_T \text{at } T \wedge p \supset \Box \varphi_k}{\vdash_T \text{at } T \wedge p \supset \Box \varphi}$$

Similar changes should be applied to rule R 8 whose second premise should now read: at $T \wedge r \supset \Box(\text{at } S \supset p) \vdash_T \text{at } T \wedge r \supset \Box(\text{after } S_0 \supset p)$.

We stated in the introduction that we provide in this paper a basis for studying temporal properties of structured programs. But a closer look both at the proof systems and completeness proofs, reveals that only one temporal operator has been used throughout the paper viz. the "$\rightsquigarrow$" operator. Consequently, knowledge of temporal logic is not needed to follow the arguments. The paper can be viewed as an effort to axiomatize the "$\rightsquigarrow$" operator. When studying this operator applied to sequential programs temporal logic can be ignored.

The situation changes when we pass to concurrent pograms. Consider for example parallel **while**-programs with shared variables. Liveness properties of such programs were studied in Owicki and Lamport (1982).

In the case of these programs various axioms and rules are not any more sound. As an illustration consider Axiom A 13:

$$\vdash_T \text{at } S \wedge p \wedge b \rightsquigarrow \text{at } S_0 \wedge p,$$

where $S \equiv$ **while** b **do** S_0 **od** is a subprogram of a parallel program T.

Now this axiom is not any more valid since the condition $p \wedge b$ does not need to hold at the moment when S is actually activated. As a result we cannot even be sure that at S_0 will eventually hold. An appropriate remedy would be to replace this axiom by the following rule:

$$\frac{\vdash_T \text{at } S \wedge p \wedge b \supset (\text{at } S \wedge p \wedge b)\, \mathcal{U}(\neg \text{ at } S)}{\vdash_T \text{at } S \wedge p \wedge b \rightsquigarrow \text{at } S_0 \wedge p}$$

whose premise states that once at $S \wedge p \wedge b$ holds the condition $p \wedge b$ remains true until the component of T containing S is once again activated. "$\mathcal{U}$" is the "until" operator of temporal logic defined as follows (see, e.g., Manna and Pnueli, 1981):

$$\models_{T,I} \varphi \mathcal{U} \psi(\sigma, i) \text{ iff } \exists k \geqslant i [\models_{T,I} \psi(\sigma, k) \wedge \forall j (i \leqslant j < k \Rightarrow \models_{T,I} \varphi(\sigma, j))].$$

Analogous modifications deal adequately with other axioms of the proof system $\mathscr{L}$. We find that corresponding axioms and rules of Owicki and Lamport (1982), do not seem to capture adequately the problem of interference of other components. For example the corresponding modification of Axiom A 13 proposed there is (essentially)

$$\vdash_T ((\text{at } S \wedge p \wedge b) \wedge \Box(\text{at } S \supset p \wedge b)) \rightsquigarrow \text{at } S_0 \wedge p,$$

but the assumption $\Box(\text{at } S \supset p \wedge b)$ seems to be too strong a requirement.

In our future work we intend to extend the results of this paper to the case of parallel **while**-programs with shared variables and CSP programs of Hoare (1978).

ACKNOWLEDGMENTS

We thank A. Pnueli for proposing the title and providing encouragement during the initial stage of this work, D. Lehmann for suggesting a simplified completeness proof of the system $\mathscr{L}$, and the referees for their comments which helped to improve the presentation.

RECEIVED October 18, 1983; ACCEPTED June 20, 1984

REFERENCES

APT, K. R. (1981), Ten Years of Hoare's Logic, a survey, part I *TOPLAS*, **3**, No. 4 431–483.

APT, K. R., AND DELPORTE C. (1983), An axiomatization of the intermittent assertion method using temporal logic (extended abstract) *in* "Proc. 10th Colloq. ICALP," Lectures Notes in Comput. Sci. Vol. 154, pp. 15–27, Springer-Verlag, Berlin/New York.

BURSTALL, R. M. (1974), Program proving as hand simulation with a little induction, *in* "Proc. IFIP" Vol. 74, pp. 308–312, North-Holland, Amsterdam.

BARRINGER, H., KUIPER, R., AND PNUELI, A. (1984), Now you may ompose Temporal Logic Specification, *in* "Proc. 16th Annu. ACM Symp. on Theory of Comput.".

GERTH, R. (1984), Transitive Logic or how to reason about temporal properties of programs in a compositional way, *in* "Proc. 16th Annu. ACM Symp. on theory of comput.".

HAREL, D. (1979), "First-Order Dynamic Logic," Lecture Notes in Comput. Sci. Vol. 68, Springer-Verlag, Berlin/New York.

HENNESSY, M. C. B., AND PLOTKIN G. D. (1979), Full abstraction for a simple programming language, *in* "Proc. 8th Sympos. MFCS," Lecture Notes in Comput. Sci. Vol. 74, pp. 108–120, Springer-Verlag, Berlin/New York.

HOARE, C. A. R. (1969), An axiomatic basis of computer programming, *Comm. ACM*, **12**, No. 10, 576–580.

HOARE, C. A. R. (1978), Communicating sequential processes, *Comm. ACM* **218**, 666–677.

LAMPORT, L. (1980), The "Hoare Logic" of concurrent programs, *Acta Inform.* **14**, No. 1, 21–37.

MANNA Z., AND PNUELI A. (1981), Verification of concurrent programs; The temporal framework, *in* "The Correctness Problem in Computer Science," Internat. Lect. Ser. in Comput Sci., Academic Press, New York/London.

MANNA Z. AND PNUELI A. (1982), Verification of concurrent programs; Temporal proof principles, *in* "Logic of Programs," Lecture Notes in Comput. Sci. Vol. 131, pp. 220–252, Springer-Verlag, Berlin/New York.

MANNA Z. AND WALDINGER R. (1978), Is "Sometime" sometimes better than "Always"?, *Comm. ACM*, **21**, No. 2, 159–172.

OWICKI S. AND LAMPORT L. (1982), Proving liveness properties of concurrent programs *TOPLAS*, **4**, No. 3, 455–495.

PNUELI, A. (1977), The temporal logic of programs, *in* "Proc. 18th Sympos. FOCS," pp. 46–57, IEEE, Providence, R.I.