Theoretical Computer Science 21 (1982) 269–279
North-Holland Publishing Company

ON THE ELIMINATION OF ITERATION QUANTIFIERS IN A FRAGMENT OF ALGORITHMIC LOGIC

J.A. BERGSTRA
Department of Computer Science, University of Leiden, 2300 RA Leiden, the Netherlands

J.-J. Ch. MEYER
Wiskundig Seminarium, Vrije Universiteit, 1081 HV Amsterdam, the Netherlands

Communicated by E. Engeler
Received July 1981

Abstract. In this paper we study the elimination of the iteration quantifier $\bigcup$ in a special set of algorithmic formulae. Something similar has been done by G. Mirkowska and E. Orlowska by means of a system of procedures. We, however, show that elimination is already possible by using the programs in our sublanguage itself.

Introduction

In this paper we shall study the elimination of the iteration quantifier $\bigcup$ in a special set of algorithmic formulae. Something similar has been done by G. Mirkowska and E. Orlowska in [4] by means of a system of procedures. However, we shall show that elimination is already possible by means of programs in our sublanguage itself.

To this end we shall now construct the special subset of the language of algorithmic logic (AL). We shall assume familiarity with the language $\mathscr{L}$ of AL as treated in [6]; so we can suffice to summarize in some detail only the syntax of $\mathscr{L}$ as far as we will need it.

1. Syntax of $\mathscr{L}$

The *alphabet* of $\mathscr{L}$ consists of the following at most enumerable sets:

V_1: the infinite set of individual variables $\{x_i \mid i \in \omega\}$,

V_0: the infinite set of propositional variables $\{a_i \mid i \in \omega\}$,

$\{\Phi_n\}_{n\in\omega}$: a family of at most enumerable sets of n-argument functors,

$\{P_n\}_{n\in\omega}$: a family of at most enumerable sets of n-argument predicates; we assume that P_2 contains the equality $=$,

{**false**, **true**, $\neg$, $\vee$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$, $\exists$, $\forall$, $\cup$, **begin** . . . ; . . . **end**, **while do** . . . **od**, **if** . . . **then** . . . **else** . . . **fi**}: the set of logical and program connectives,

{(,), [,], /}: the set of auxiliary symbols.

The sets of *well-formed* expressions of $\mathscr{L}$ are the following:

T: the set of classical terms τ,

F_0: the set of open classical formulae, which we shall denote by α, β, γ, δ,

S: the set of substitutions $s = [x_{k_1}/\tau_1, \ldots, x_{k_n}/\tau_n, a_{l_1}/\alpha_1, \ldots, a_{l_m}/\alpha_m]$, in which $x_{k_i} \in V_1$, $a_{l_i} \in V_0$, $\tau_i \in T$ and $\alpha_i \in F_0$,

FS: the set of programs, defined as the least set of expressions closed under:

(FS$_1$) every $s \in S$ is an element of FS;

(FS$_2$) if $\alpha \in F_0$ and $K, M \in$ FS, then also the expressions **begin** K; M **end** and **if** α **then** K **else** M **fi** are elements of FS;

(FS$_3$) if $\alpha \in F_0$, and $K \in$ FS, then also **while** α **do** K **od** is an element of FS.

FS$_0$: the least subset of FS closed under (FS$_1$) and (FS$_2$): the loop-free programs in FS.

We shall use the letters K, L, M and P for programs $\in$ FS.

We shall also make use of the following abbreviations:

(1) **if** α **then** K **fi** instead of
 if α **then** K **else** [ ] **fi**, where [ ] is a dummy substitution;

(2) **begin** K_1; K_2; ... ; K_n **end** instead of
 begin ··· **begin** K_1; K_2 **end**; ... K_n **end**

(3) **while** α **do** K_1; ... ; K_n **od** instead of
 while α **do begin** K_1; ... ; K_n **end od**

(4) **if** α **then** K_1; ... ; K_n **else** K_{n+1}; ... ; K_m **fi** instead of
 if α **then begin** K_1; ... ; K_n **end else begin** K_{n+1}; ... ; K_m **end fi**

(5) K^i instead of **begin** $\underbrace{K; K; \ldots; K}_{i \text{ times}}$ **end**

F: the set of formulae of AL, defined as the least set of expressions closed under:

(F_1) $V_0 \cup \{\textbf{true}, \textbf{false}\} \subset F$;

(F_2) if $\rho \in P_n (n \geqslant 0)$ and $\tau_1, \ldots \tau_n \in T$, then $\rho(\tau_1, \ldots, \tau_n) \in F$;

(F_3) if $\varphi, \chi \in F$, then $\neg\varphi$, $(\varphi \vee \chi)$, $(\varphi \wedge \chi)$, $(\varphi \Rightarrow \chi)$ and $(\varphi \Leftrightarrow \chi)$ are $\in F$;

(F_4) if $K \in$ FS and $\varphi \in F$, then $K\varphi$, $\bigcup K\varphi$ and $\bigcap K\varphi$ are formulae $\in F$.

We will consider the following fragment F^* of F, defined as the least set of expressions closed under:

(F_1^*) $F_0 \subset F^*$;

(F_2^*) if $K \in$ FS and $\varphi \in F^*$, then $K\varphi$, $\bigcup K\varphi \in F^*$;

(F_3^*) if $\varphi, \chi \in F^*$, then also $(\varphi \wedge \chi)$, $(\varphi \vee \chi) \in F^*$.

We shall use the Greek letters φ, χ, ψ and θ for formulae $\in F^*$.

2. Semantics of $\mathscr{L}$

The semantics is the usual of AL (see [3, 5, 6]):

By a given valuation v of the elements of $V_0 \cup V_1$ the values $\tau_R(v)$ of elements $\tau \in T$ and truth values $\alpha_R(v)$ of elements $\alpha \in F_0$ are determined in the usual way;

and the realizations of formulae $\in F^*$ are defined as

$$(K\varphi)_R(v) = \begin{cases} \varphi_R(K_R(v)) & \text{if } K_R(v) \text{ is defined,} \\ \text{false} & \text{otherwise} \end{cases}$$

and

$$(\bigcup K\varphi)_R(v) = \sup_{i\in\omega}(K^i\varphi)_R(v).$$

Let W be the set of all valuations of elements of $V_0 \cup V_1$, then we define AL$\models \varphi$ (with $\varphi \in F$) as $\forall v \in W$: $\varphi_R(v)$ is true.

Now we shall state and prove the following

Theorem. *Every $\varphi \in F^*$ is equivalent with some $\psi \in F^*$ s.t. $\psi = K$* **true** *for a certain $K \in$* FS.

Corollary. *So by this theorem it is possible to eliminate the iteration quantifier $\bigcup$ in our subset F^* of* AL.

Proof of the theorem. With induction on the complexity of φ.

The proof is heavily based upon the following normal form theorem, the proof of which can be found in [1, 2]:

For every program $K \in$ FS, there exists a program $\tilde{K}$ of the following form:

begin s; **while** α **do** M **od end**

with $s \in S$, $\alpha \in F_0$ and $M \in FS_0$, such that for every realisation R and for every valuation v, $K_R(v)$ is defined iff $\tilde{K}_R(v)$ is defined, and if they are defined then, for all variables $x \in (V_0 \cup V_1)\backslash(V(\tilde{K})\backslash V(K))$, $K_R(v)(x) = \tilde{K}_R(v)(x)$.

($V(K)$ stands for the set of variables occurring in $K \in$ FS; hence $V(\tilde{K})\backslash V(K)$ is the set of auxiliary variables occurring only in $\tilde{K}$ and not in K already.)

We shall prove our theorem with induction on the complexity of φ. Parts (0) thru (iii) are fairly straightforward, part (iv) is far less easy.

(0) For a φ with the lowest complexity, i.e. $\varphi \in F_0$, take $\psi = K'$ **true** with K': **while** $\neg\varphi$ **do** [ ] **od**.

Then AL$\models \varphi \Leftrightarrow \psi = K'$ **true**

(i) If $\varphi = \varphi_1 \wedge \varphi_2$ with φ_1, $\varphi_2 \in F^*$, then φ_1 and φ_2 are of lower complexity, so by the induction hypothesis:

AL$\models \varphi_1 \Leftrightarrow K_1$ **true** for a certain $K_1 \in$ FS and

AL$\models \varphi_2 \Leftrightarrow K_2$ **true** for a certain $K_2 \in$ FS.

So AL$\models \varphi \Leftrightarrow K_1$ **true** $\wedge K_2$ **true**.

Notice that K_1 only can influence a finite number of values of x_i and a_i; suppose that these are $x = \{x_{k_1}, \ldots, x_{k_n}\}$ and $a = \{a_{l_1}, \ldots, a_{l_m}\}$ respectively.

Therefore we can reserve variables of the infinite set $V_0 \cup V_1$, which we can assign the original values of the x_i and a_i, and which are not affected by K_1. Name these variables $y = \{x_{p_1}, \ldots, x_{p_n}\}$ and $b = \{a_{q_1}, \ldots, a_{q_m}\}$ respectively.

Let K' now be the program: **begin** $[y/x]$; $[b/a]$; K_1; $(K_2)_{y,b}$ **end** with $(K_2)_{y,b}$ is K_2 in which all occurrences of the x_i and a_i have been replaced by the corresponding y_i and b_i respectively.

Then AL $\models \varphi \Leftrightarrow K'$ **true**.

So take $\psi = K'$ **true**.

(ii) If $\varphi = \varphi_1 \vee \varphi_2$ with $\varphi_1, \varphi_2 \in F^*$, then by the induction hypothesis:

AL$\models \varphi_1 \Leftrightarrow K_1$ **true** for a certain $K_1 \in FS$ and

AL$\models \varphi_2 \Leftrightarrow K_2$ **true** for a certain $K_2 \in FS$.

So AL$\models \varphi \Leftrightarrow K_1$ **true** $\vee$ K_2 **true**.

By the normal form theorem [1, 2] it is no restriction to assume:

K_1: **begin** s_1;
    while α_1 **do** M_1 **od**
  end

and

K_2: **begin** s_2;
    while α_2 **do** M_2 **od**
  end

respectively, for some $s_1, s_2 \in S$, $\alpha_1, \alpha_2 \in F_0$, $M_1, M_2 \in$ FS.

Analogously to (i), take variables $y \subset V_1$ and $b \subset V_0$ s.t. the programs s_1 and M_1 do not affect their valuations, to copy x and a, of which, the valuations are affected by s_1 or M_1.

Let K' now be the program:

begin s_1; $(s_2)_{y,b}$;
  while $\alpha_1 \wedge (\alpha_2)_{y,b}$ **do** M_1; $(M_2)_{y,b}$ **od**
end,

in which $\varphi_{y,b}$ is defined as φ in which every occurrence of x_i and a_i has been replaced by the corresponding y_i and b_i respectively.

Now AL$\models \varphi \Leftrightarrow K'$ **true**.

So take $\psi = K'$ **true**.

(iii) If $\varphi = K\varphi_1$ with $K \in$ FS and $\varphi_1 \in F^*$, then φ_1 is of a lower complexity and so: AL$\models \varphi_1 \Leftrightarrow K_1$ **true** for a certain $K_1 \in$ FS.

So AL$\models \varphi \Leftrightarrow KK_1$ **true**.

Take K': **begin** K; K_1 **end**, then AL$\models \varphi \Leftrightarrow K'$ **true**.

So take $\psi = K'$ **true**.

(iv) Suppose $\varphi = \bigcup K\varphi_1$ with $K \in$ FS and $\varphi_1 \in F^*$. By the induction hypothesis it holds that AL$\models \varphi_1 \Leftrightarrow L$ **true** for some $L \in$ FS.

So: $\varphi_1 \wedge \neg\beta \Leftrightarrow (L$ **true**$) \wedge \neg\beta \Leftrightarrow \tilde{L}$ **true** where $\tilde{L}$: **begin while** β **do** [ ] **od**; L **end**.

Thus AL$\models \bigcup K\varphi_1 \Leftrightarrow \bigcup K(\tilde{L}$ **true**$)$.

K and $\tilde{L}$ may contain many while-statements. As this is a little difficult to deal with in combination with the iteration quantifier $\bigcup$, we shall use the normal form theorem first to be able to restrict ourselves to statements K and $\tilde{L}$ with only *one* while statement.

Then we shall use the power of the iteration quantifier to give an equivalent formula containing exactly *one* while-statement, which will occur in a subformula of the form (M **true**).

By the normal form theorem K and $\tilde{L}$ can be assumed to have the forms:

K: **begin** s; **while** β **do** K' **od end**,

$\tilde{L}$: **begin** s_0; **while** γ **do** L' **od end**,

for certain $s, s_0 \in S$, $\beta, \gamma \in F_0$ and $K', L' \in FS_0$.

Next we notice that $\bigcup K(\tilde{L}\ \mathbf{true})$ is equivalent with

$$s_1 \bigcup K_1((\tilde{L}\ \mathbf{true}) \wedge \neg\beta)$$

in which

s_1: $[b/\mathbf{true}]$ and

K_1: **begin if** $\neg\beta \vee b$ **then** s; $[b/\mathbf{false}]$**fi**;
if β **then** K' **fi**
end

where b is a variable $\in V_0$ which is not affected by K' and s.

Consequently $AL \models \bigcup K\varphi_1 \Leftrightarrow s_1 \cup K_1(\tilde{L}\ \mathbf{true}) \Leftrightarrow s_1 \cup K_1$ (**begin** s_0; **while** γ **do** L' **od end true**).

Finally we construct a program P s.t. $AL \models s_1 \cup K_1$ (**begin** s_0; **while** γ **do** L' **od end true**)$\Leftrightarrow P$ **true**, i.e. P terminates iff the program $\tilde{L}$ terminates after applying first s_1 and after that K_1 several times.

We can consider P as a search program. To find a state of termination of $\tilde{L}$, P must first execute s_1 and then it must investigate all applications of $\tilde{L}$ to iterations of K_1 in a constructive, parallel way.

The problems that arise here, are the following: P cannot remember all the things we have investigated already, for in a program we can affect only a finite number of x_i and b_i, and the program $\tilde{L}$ need not terminate, which implies infinitely many investigations to be remembered.

Furthermore, for the use in P we have no possibility of access to x_i, b_i by means of a pointer i which can be moved, nor have we at our disposal any counters which register for instance how many iterations of K_1 we have done already in our investigation. So P has to do much extra work (viz. repeats of investigations already done) to organize the search with tests of identity only.

This entails a further difficulty: the iteration of K_1 in P acts as a sort of clock (indicator of progress) in the work of searching. This iteration, however, can get into a loop and in that case cannot act as a clock any longer.

So we must regularly test whether this iteration has got into a loop or not, and if so, we must take another clock. Then we take as clock the execution (in steps) of $\tilde{L}$ on the initial valuation v_0.

But this indicator can get into a loop too and in that case we consider $K(v_0)$ as the initial valuation from that moment onwards and repeat the preceding.

We shall now state the program after first explaining the meaning of 'auxiliary' variables that we shall use.

Notice first that the programs s_1, K_1, s_0 and L' can affect only finitely many x_i of the infinite set $V_1 = \{x_i \mid i \in \omega\}$ and finitely many a_i of the infinite set $V_0 = \{a_i \mid i \in \omega\}$. Suppose that all of these programs affect the valuation of a subset of $\{x_0, \ldots, x_{n-1}\}$ and a subset of $\{a_0, \ldots, a_{m-1}\}$ only. In P we need some copies of values of $\{x_0, \ldots, x_{n-1}, a_0, \ldots, a_{m-1}\}$, which we place in the variables $\{x_{kn}, \ldots, x_{(k+1)n-1}, a_{km}, \ldots, a_{(k+1)m-1}\}$ for $k \in \{1, 2, \ldots, 7\}$ of which s_1, K_1, s_0 and L' do not affect the valuations anyway.

We shall abbreviate $\{x_{kn}, \ldots, x_{(k+1)n-1}, a_{km}, \ldots, a_{(k+1)m-1}\}$ $(k \in \{0, 1, 2, \ldots, 7\})$ as:

for

$k = 0$: $\mathscr{x}$,

$k = 1$: $\mathscr{x}0$, in which we shall place a copy of the initial values of $\mathscr{x}$ after applying s_1,

$k = 2$: $\mathscr{t}$,

$k = 3$: $\mathscr{clock1}$ in which we shall place the first clock,

$k = 4$: $\mathscr{w}$,

$k = 5$: $\mathscr{y}$,

$k = 6$: $\mathscr{z}$,

$k = 7$: $\mathscr{clock2}$ in which we shall place the second, third etc. 'clock'.

Furthermore, we need some 'extra' propositional variables $\in V_0$ outside the set $\{a_0, a_1, \ldots, a_{m-1}\}$ that we shall name

b_1 (which, if it has valuation true, will indicate that there is no loop yet in the first clock),

b_2 (which, if it has valuation true, will indicate that there is no loop yet in the second, third etc. clock),

c (which, if it has valuation false, will indicate that a termination point of $\tilde{L}$ has been found), and

b_3.

Also, to be able to apply s_0, s_1, K, L', γ to the copies, we use modified versions of s_0, s_1, K, L', γ. For example, for being able to apply K to the values of $\mathscr{y}$, we use $K_{\mathscr{y}}$ which is the program K in which all occurrences of x_i and a_i have been replaced by the corresponding x_{5n+i} and a_{5n+i} respectively. Note that $K_{\mathscr{y}}$ only affects the valuation of $\mathscr{y}$ (just as K only affects the valuation of $\mathscr{x}$), so applying $K_{\mathscr{y}}$ does not change the current values of $\mathscr{x}$, $\mathscr{clock1}$, $\mathscr{clock2}$, $\mathscr{v}$, $\mathscr{w}$, $\mathscr{z}$. Other modified versions of γ, s_0, s_1, K, L' are defined and denoted analogously.

The program P is now

begin N_1; N_2; N_3 **end**

where N_1 stands for

```
begin s1;                                   - first execute s1
  [x0/x];                                   - store the initial values of x in x0
  [clock1/x];                               - and in clock1
  (K1)clock1; [c/true];                     - execute K1 on clock1 to get its new values
  [b1/(x0 ≠ clock1)];                       - no t.p.¹ has been found yet
                                            - no loop in the new clock1 yet?
  while b1 ∧ c                              - while clock1 loop-free and no t.p. found:
  do (K1)clock1;                            - reset clock1
     [x/x0];                                - set x back to its initial values
     while (x ≠ clock1) ∧ c                 - while x ≠ clock1 and no t.p. found:
     do [y/x];                              - store the current values of x in y
        (s0)y;                              - execute s0 on y
        [z/x0];                             - store the current values of x0 in z
        while (z ≠ clock1) ∧ γy ∧ c         - while z is not yet clock1 and γ is true for y,
                                              and no t.p. found:
        do L'y;                             - execute L' on y and
           (K1)z;                           - K1 on z to obtain next z
        od;
        if ¬γy then [c/false] fi;           - if a y has been found for which ¬γ
                                              a t.p. has been found!
        K1                                  - execute K1 on x to get the next values of x
     od;
     Q1                                     - check if clock1 is still loop-free
  od
end
in which Q1:
begin [l/x0];                               - initialize l to x0 and ω to clock1
  [ω/clock1];
  (K1)ω;                                    - execute K1 on ω to get the next clock1-values
                                              in ω
  while (l ≠ clock1) ∧ c                    - while l not yet clock1 and no t.p. found:
  do [b1/(b1 ∧ (l ≠ ω))];                   - check if the old clock1-values l still differ
                                              from the next clock1-candidate
     (K1)l                                  - execute K1 on l to get next ex-clock1-values
  od;
  [b1/(b1 ∧ (clock1 ≠ ω))]                  - is the current clock1 ≠ the next clock1?
end                                         - note that clock1 is not affected by Q1!
```

Comment (see also Fig. 1). N_1 is the part of P in which the iteration of K_1 can act as 'clock' (i.e. indicator of progress of investigation). The iteration is performed on v_0, affecting the values of *clock1*. For each valuation

$$v_i = (K^i_{1,clock1})_R(v_0)$$

[1] t.p. stands for termination point of $\tilde{L}$.

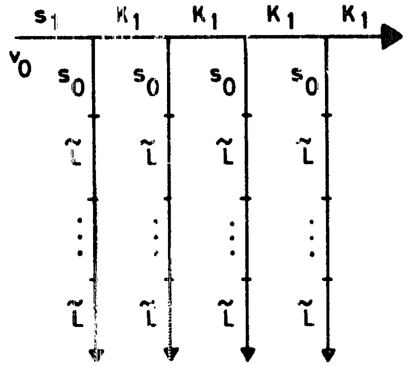

Fig. 1. $\mathit{clock1}$ is still loop-free.

with $i \geq 0$ (in which therefore

$$v_i(\mathit{clock1}) = \mathit{clock1}_R(v_i) = x_R((K_1^i)_R(v_0))),$$

N_i tests i steps of the program $\tilde{L}$ applied to all v such that

$$v = (K_1^j)_R(v_0) \quad \text{with } 0 \leq j < i$$

on termination (it does so by means of a copy y), and then it verifies by means of Q_1, whether iteration of K_1 is still loop-free (this is done by checking if the next iteration generates a valuation of $\mathit{clock1}$ that we have met with already) and thus can still act as indicator for the investigation of the next iteration of $(K_1)_{\mathit{clock1}}$.

If this is the case N_1 proceeds to v_{i+1}, otherwise N_1 terminates and hands over the search to program N_2.

N_2:

begin while $(x0 \neq \mathit{clock1}) \wedge c$	– while $x0$ not yet $\mathit{clock1}$ and to t.p. found:
do $[\mathit{clock2}/x0]$;	– set $\mathit{clock2}$ to $x0$
$[b_2/\mathbf{true}]$;	– no loop yet in $\mathit{clock2}$
while $b_2 \wedge c$	– while no loop in $\mathit{clock2}$ and no t.p. found:
do if $\mathit{clock2} = x0$ **then** $(s_0)_{\mathit{clock2}}$	– the first time reset $\mathit{clock2}$ by means of s_0,
else L'_{clock2} **fi**;	– otherwise by means of L'
if ${}^1\gamma_{\mathit{clock2}}$ **then** $[c/\mathbf{false}]$ **fi**;	– if ${}^1\gamma$ for $\mathit{clock2}$: t.p. is found
K_1;	– execute K_1 on x;
$[b_3/(x = \mathit{clock1})]$;	– Is $x = \mathit{clock1}$ for the first time?
while $((x \neq \mathit{clock1}) \vee b_3) \wedge c$	– while $x \neq \mathit{clock1}$ or $[x = \mathit{clock1}$ for the first time$]$ and no t.p. found[2]
do $[y/x]$;	– initialize auxiliary y to x
$(s_0)_y$; $(s_0)_{x0}$; $[z/x0]$;	– execute s_0 on y and $x0$; set z to $x0$
while $(z \neq \mathit{clock2}) \wedge \gamma_y c$	– while $z \neq \mathit{clock2}$
do L'_y; L'_z **od**;	– execute L' on y and z
if ${}^1\gamma_y$ **then** $[c/\mathbf{false}]$ **fi**;	– has a t.p. been found?

[2] b_3 is true indicates: you must enter the loop just once again

if $x \neq \mathit{clock1}$ **then** K_1 **fi**;	– if $x \neq \mathit{clock1}$ reset x by means of K_1
$[b_3/((\neg b_3) \wedge (x = \mathit{clock1}))]$	– Is $x = \mathit{clock1}$ for the first time?
od;	
Q_2	– Is $\mathit{clock2}$ still loop free?
od;	
$(K_1)_{x0}$	– reset $x0$
od	
end	
in which Q_2:	
begin $[\ell/x0]$;	– initialize ℓ to $x0$
$(s_0)_\ell$;	– execute s_0 on ℓ
$[\omega/\mathit{clock2}]$; L'_ω;	– initialize ω to $\mathit{clock2}$ – execute L' on ω to get next $\mathit{clock2}$-value
while $(\ell \neq \mathit{clock2}) \wedge c$	
do $[b_2/(b_2 \wedge (\ell \neq \omega))]$;	– is ℓ still $\neq$ the next $\mathit{clock2}$-candidate?
L'_ℓ	– reset ℓ
od;	
$[b_2/(b_2 \wedge (\mathit{clock2} \neq \omega))]$	– is the current $\mathit{clock2} \neq$ the next $\mathit{clock2}$?
end	– note that Q_2 does not affect $x0$ and $\mathit{clock2}$
N_3:	
begin $(s_0)_{\mathit{clock1}}$;	– execute s_0 on $\mathit{clock1}$
while $\gamma_{\mathit{clock1}} \wedge c$	– while γ holds for $\mathit{clock1}$ and if no previous t.p. has been found by N_1 or N_2:
do L'_{clock1} **od**	– execute L' on $\mathit{clock1}$
end	

Comment. N_2 is the part of P in which the execution of $\tilde{L}$ on 'initial' values $v'_0(x0)$, which takes place with the aid of variables $\mathit{clock2}$ (and via a valuation v''_0 such that

$$v''_0(\mathit{clock2}) = v'_0(x0) \quad \text{and}$$

$$v''_0(x) = v'_0(x) \quad \text{for all } x \in V_1 \backslash \mathit{clock2}),$$

functions as clock.

(Initially $v''_0 = v'_0 = v_0$; if $\tilde{L}(v_0)$, however, cannot operate as clock any longer, N_2 takes $\tilde{L}(v'_0)$ with $v'_0 = (K_1)_{x0}(v_0)$ as clock, etc., as long as it holds for the valuation v'_0 that $v'_0(x0) \neq v'_0(\mathit{clock1})$; see Figs. 2–4.)

For notational convenience we define $M^{(k)}$ as the first k steps of program M.

For each valuation

$$v''_k = \tilde{L}^{(k)}_{\mathit{clock2}}(v''_0) \quad \text{with } k > 0$$

(in which thus

$$v''_k(\mathit{clock2}) = x0_R((\tilde{L}^{(k)}_{x0}(v'_0)) = x_R(\tilde{L}^{(k)}(K^{i_0}(v_0))) \quad \text{for some } i_0 \geq 0),$$

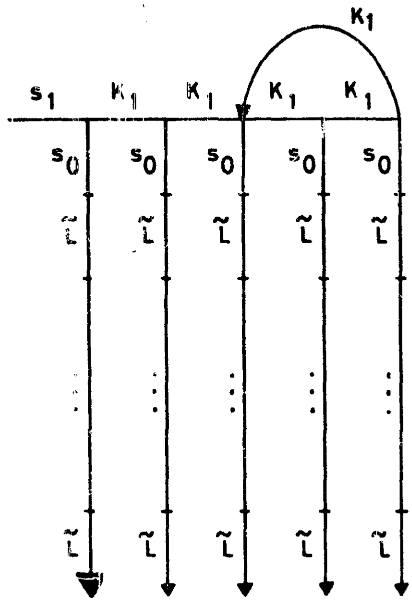

Fig. 2. *clock2* takes over.

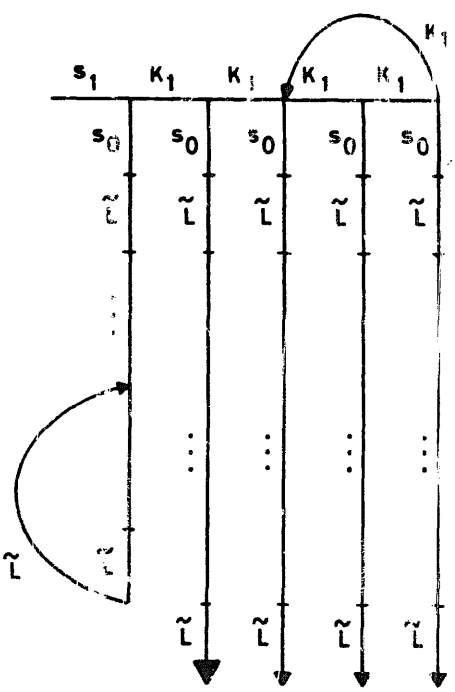

Fig. 3. A new *clock2* has been taken.

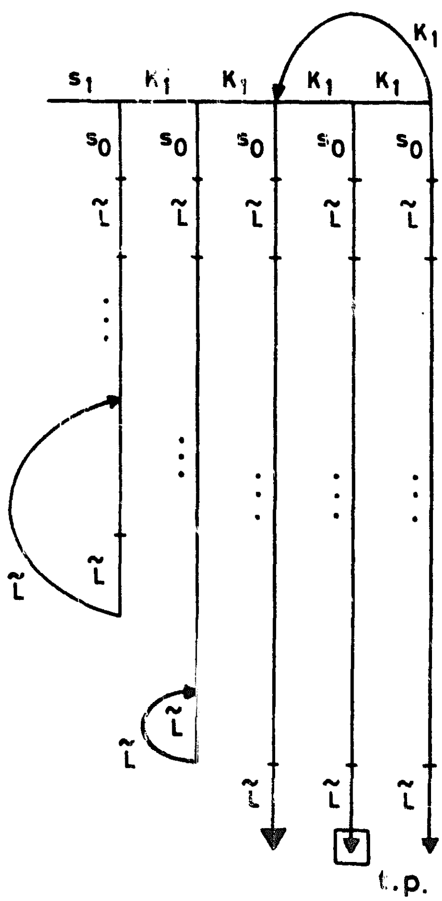

Fig. 4. A termination point has been found.

N_2 investigates k steps of the program $\tilde{L}_{x0}$ applied to all v with

$$v=(K^{j}_{1,x0})_R(v'_0)\quad \text{for } 0\leq j<i-i_0,$$

with i such that v_i was the 'indicator' valuation after termination of N_1, on termination. (This investigation takes place by means of a copy y.)

Then it verifies by means of Q_2 whether the clock $\tilde{L}(v'_0)$ is still loop-free or not. If so, N_2 proceeds to v''_{k+1}. If not, then N_2 considers $(K_1)_{x0}(v''_0)$ as the new initial valuation v'_0 and $(K_1)_{clock1}((K_1)_{x0}(v''_0))$ as the new v''_0, etc., until $v'_0(x0)=v'_0(clock1)$. Then the 'initial' values of $x0$ are the same as those of $clock1$, which were the last values to be checked of the former clock from N_1 when N_1 terminated, and N_3 takes over to investigate the termination of $\tilde{L}$ on these values.

Having given the program P with some explanatory comments, we have completed the proof of our theorem. □

References

[1] L. Banachowski, Investigations of properties of programs by means of the extended algorithmic logic I, *Fund. Inform.* **1** (1) (1977) 93–119.

[2] A. Kreczmar, Effectivity problems of algorithmic logic, *Fund. Inform.* **1** (1) (1977) 19–32.

[3] G. Mirkowska, Algorithmic logic and its applications in the theory of programs I, *Fund. Inform.* **1** (1) (1977) 1–17.

[4] G. Mirkowska and E. Orlowska, An elimination of iteration quantifiers in a certain class of algorithmic formulas, *Fund. Inform.* **1** (3) (1978) 347–355.

[5] H. Rasiowa and R. Sikorski, *Mathematics of Metamathematics*, Monografie Mat. **41** (Polish Scientific Publishers, Warsaw, 1963).

[6] A. Salwicki, Formalized algorithmic languages, *Bull. Acad. Polon. Sci., Sér. Math. Astronom. Phys.* **18** (1970) 227–232.