Department of Applied Analysis
and Computer Science
Technical Report CS-73-04
January, 1973

A NEW PROOF OF THE UNDECIDABILITY RESULTS FOR FLOWCHART SCHEMAS

by
E.A. Ashcroft
University of Waterloo

(Revised June 1974)

Department of Applied Analysis
and Computer Science
Technical Report CS-73-04
January, 1973

A NEW PROOF OF THE UNDECIDABILITY RESULTS FOR FLOWCHART SCHEMAS

by

E.A. Ashcroft
University of Waterloo

(Revised June 1974)

A NEW PROOF OF THE UNDECIDABILITY RESULTS FOR FLOWCHART SCHEMAS

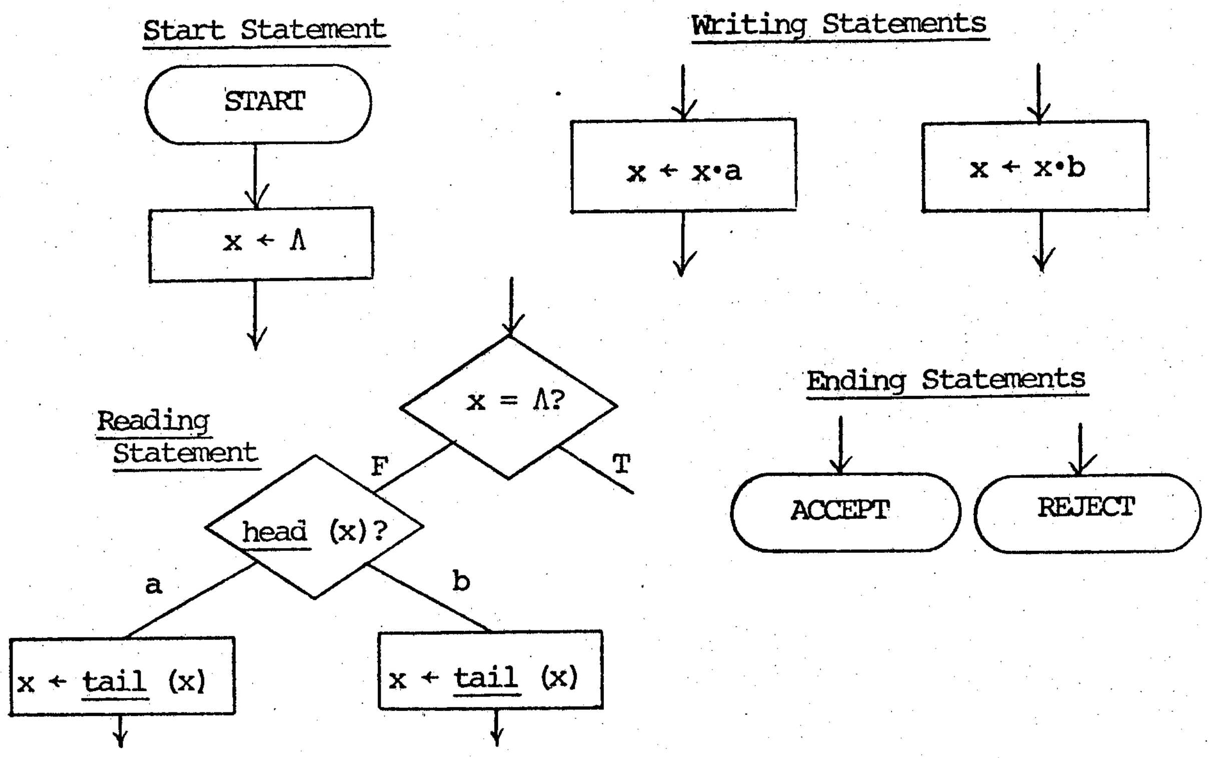
E. A. Ashcroft
Computer Science
University of Waterloo
Waterloo, Ontario
Canada

A NEW PROOF OF THE UNDECIDABILITY RESULTS FOR FLOWCHART SCHEMAS

The usual proof that divergence and non-halting are not partially decidable for flowchart schemas [1] uses a two-stage simulation - a simulation of Turing machines by two-headed automata (with infinite tapes) and then a simulation of two headed automata by flowchart schemas. In the following proof we use one simulation of Post machines by flowchart schemas.

We shall consider Post machines starting with empty tape. These machines are related to Post normal systems [2] but are actually the Single-Register Machines of Shepherdson and Sturgis [3].

We will take a binary Post machine P (over alphabet {a, b}) to be a flowchart constructed from the following components (the generalisation to larger alphabets is obvious):



Λ	denotes the empty list or queue.
$x \cdot a$	denotes the queue x with 'a' added on the right.
$x \cdot b$	denotes the queue x with 'b' added on the right.
$\underline{\text{head}}(x)$	denotes the left most symbol of x, if x is non-empty.
$\underline{\text{tail}}(x)$	denotes x without its left most symbol, if x is non-empty.

We will say that a Post machine is in <u>normal form</u> if it never takes the T branch of any of its $x = \Lambda$ tests.

In [3] it is shown that binary Post machines can compute all partial recursive functions, and therefore are as general as Turing machines. It is also mentioned there that binary Post machines in normal form are as general as arbitrary Post machines (we prove this in the appendix, for completeness).

Since binary normal form Post machines are as general as Turing machines, <u>non-halting of such machines is not partially decidable</u>.

We shall show how to construct, from each binary Post machine P in normal form, flowchart schemas S_{P_1} and S_{P_2} such that

S_{P_1} diverges if and only if P does not halt

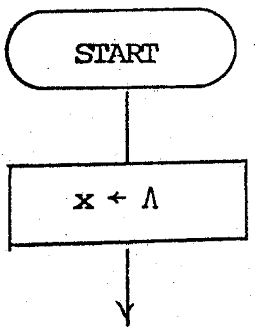
S_{P_2} does not halt if and only if P does not halt.

S_{P_1} and S_{P_2} use two variables ℓ and r, which 'keep track of' the left and right hand ends respectively of the queue x. In general, when the length of x is k, and the value of ℓ is y, then the value of r is $f^k(y)$, and the sequence $\langle p(y), p(f(y)), p(f^2(y)), \ldots, p(f^{k-1}(y))\rangle$, is a coded version of x : T represents a, and F represents b.

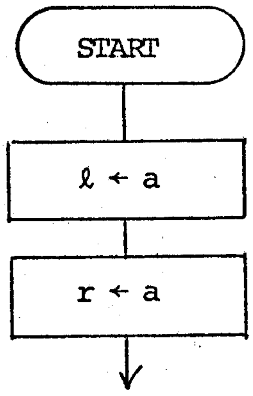
Note that in a normal form Post machine we can omit the $x = \Lambda$ tests.

<u>Construction of S_{P_1}</u>

<u>Start statement fragments:</u>

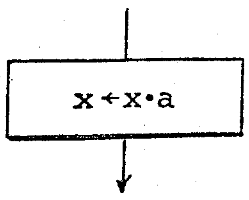


in P becomes

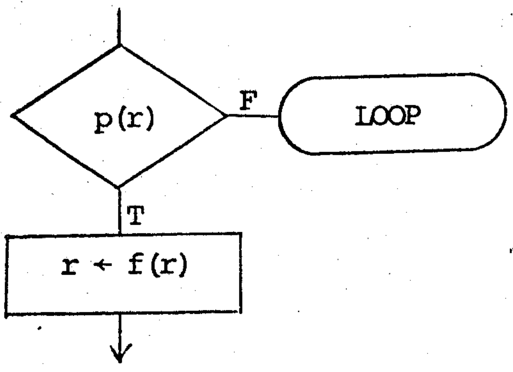


in S_{P_1}

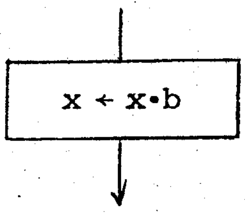
<u>Writing statement fragments:</u>



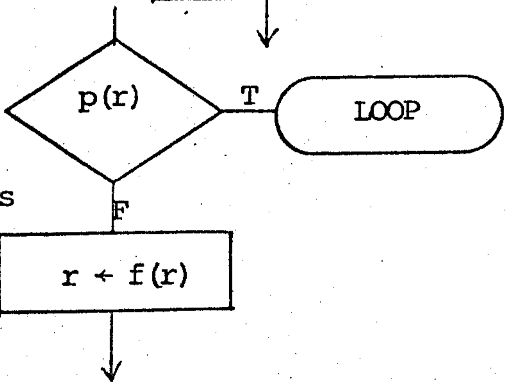
in P becomes



in S_{P_1}

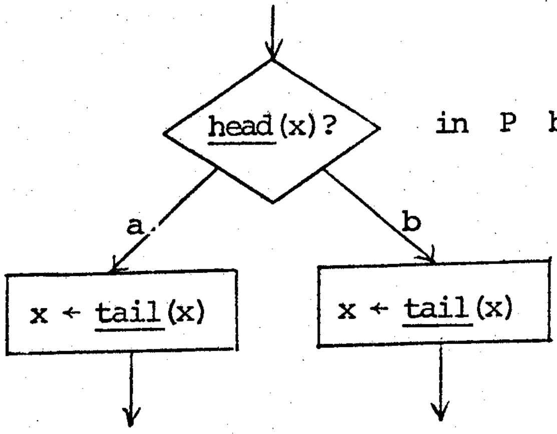


in P becomes

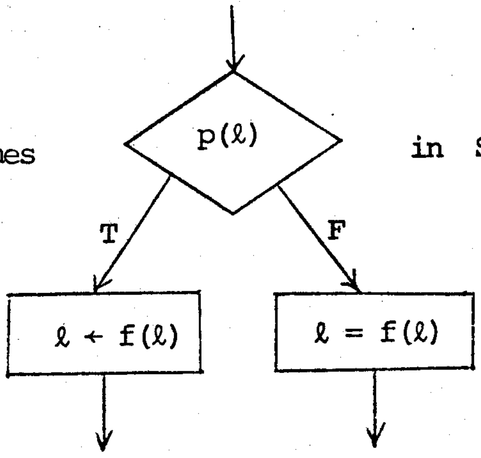


in S_{P_1}

<u>Reading statement fragments:</u> (no $x = \Lambda$ test is required since P is in normal form)

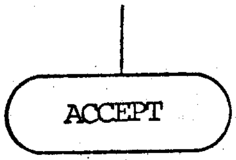


in P becomes

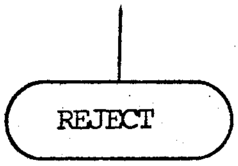


in S_{P_1}

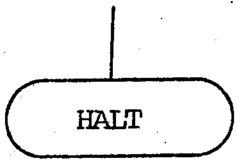
<u>Ending fragments:</u>



and



in P become



in S_{P_1} .

We consider these fragments as elementary pieces of P and S_{P_1} .

<u>Theorem 1.</u> S_{P_1} diverges if and only if P does not halt.

<u>Proof</u> Consider any interpretation for which computation of S_{P_1} does not get to LOOP. We claim that the computation of S_{P_1} follows the same path as the computation of P , and at each step, i.e. between fragments, the values of x , ℓ and r are related as follows:

If $x = \alpha_1\alpha_2 \ldots \alpha_n$ in P (i.e., the length of x is n), then

$$r = f^n(\ell) \quad \text{and} \quad \text{for } 1 \leq k \leq n$$

$$\alpha_k = a \iff p(f^{k-1}(\ell)) = T$$

$$\alpha_k = b \iff p(f^{k-1}(\ell)) = F$$

This is clearly true after the Start statement fragments when $x = \Lambda$, $n = 0$ and $\ell = r = a$.

To check that the relationship is preserved by the reading and writing fragments, let $x = \alpha_1\alpha_2 \ldots \alpha_n$, ℓ and r be the values before the fragment, and $x' = \alpha_1'\alpha_2' \ldots \alpha_m'$, ℓ' and r' be the values after the fragment. For writing statement fragments, $m = n+1$, $\ell' = \ell$, $\alpha_k' = \alpha_k$ for $1 \leq k \leq n$, and $r' = f(r) = f^{n+1}(\ell) = f^m(\ell')$. Note that $p(f^{m-1}(\ell')) = p(f^n(\ell)) = p(r)$; this value corresponds to the symbol added (α_m') by the restriction that the computation does not get to LOOP. For reading statements, since P is in normal form, $n > 0$, $m = n-1$, $\alpha_k' = \alpha_{k+1}$ for $1 \leq k \leq m$, $r' = r$ and $\ell' = f(\ell)$. Thus $r' = r = f^n(\ell)$ $= f^{m+1}(\ell) = f^m(\ell')$. Note that for $1 \leq k \leq m$, $p(f^{k-1}(\ell')) = p(f^k(\ell))$ $= p(f^{(k+1)-1}(\ell))$; so the values of p for α_k' are correct, since they were correct for α_{k+1} . It simply remains to show that the same exits are taken from the reading statement fragments, which follows immediately from the correspondence between α_1 and the value of $p(\ell)$.

We also claim that there is always some interpretation that does not get to LOOP. This follows because the tests that lead to LOOP are all free tests.

Hence there is an interpretation for which S_{P_1} halts if and only if the computation of P halts. i.e. S_{P_1} diverges if and only if P does not halt. □

Construction of S_{P_2}

The construction of S_{P_2} is similar to that of S_{P_1} but with LOOP replaced by HALT.

Theorem 2. S_{P_2} does not halt if and only if P does not halt.

Proof The reasoning of the proof of Theorem 1 applies, in particular the relationship between x , ℓ and r for interpretations that do not hit the HALT's (that were LOOP's in S_{P_1}) is the same as before. Hence there is an interpretation for which S_{P_2} does not halt if and only if the computation of P does not halt. $\square$

Since P is an arbitrary binary Post machine in normal form, and for this class of machines non-halting is not partially decidable, the two theorems give us our undecidability results:

(1) Divergence of flowchart schemas is not partially decidable.

(2) Non-halting of flowchart schemas is not partially decidable.

Comments

i) The class of schemas for which the undecidability results are proved is exactly the same class $\mathcal{S}_2$ as is used in the 'classical' proof in [1].

ii) The motivation for this new proof was to popularise the use of Post machines in proving results in program schema theory, since they seem to be particularly suitable for this. Another 'natural' technique, the use of the Post Correspondence Problem, could also have been used, quite elegantly, to prove (1) but not (2).

References

[1] Luckham, D., Park, D. & Paterson, M. "On Formalized Computer Programs". Journal of Comput. & System Sci., Vol. 4, No. 3, 1970.

[2] Post, E. L. "Formal reductions of the general combinatorial decision problem". Amer. J. Math. 65, 197-215.

[3] Shepherdson, J. C. & Sturgis, H. E. "Computability of Recursive Functions". J. Assoc. Comput. Math., Vol. 10, 217-255.

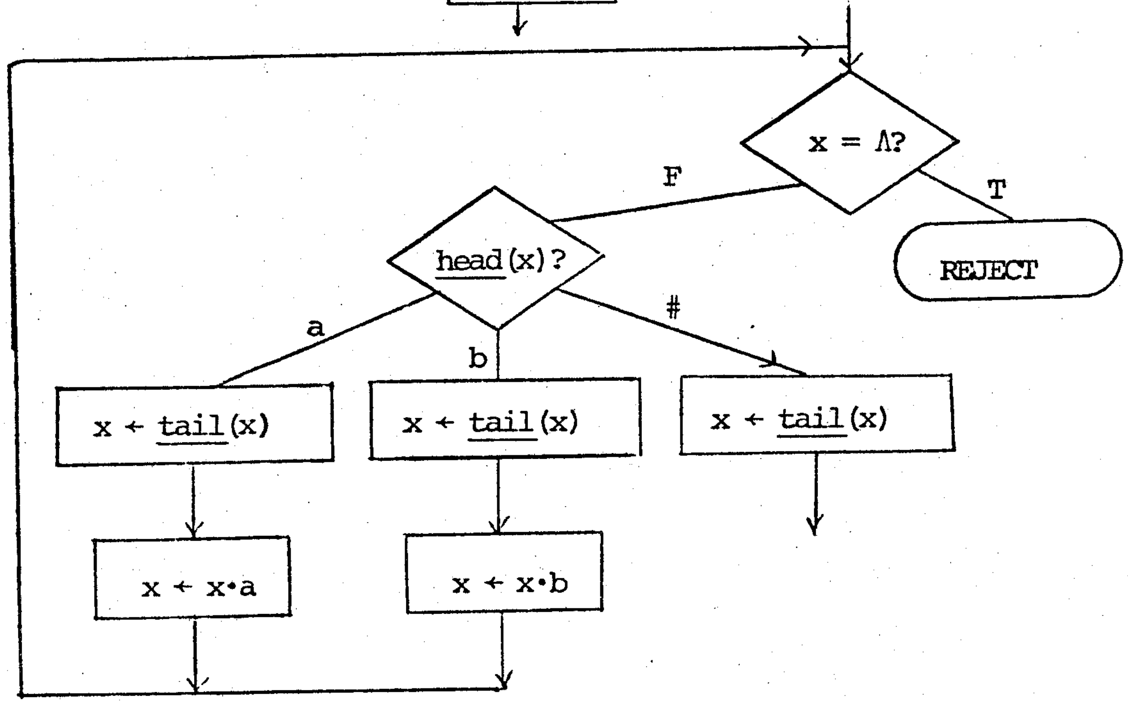
APPENDIX: Binary normal-form Post Machines are as general as arbitrary binary Post machines

We will show that binary Post machines can be simulated by ternary Post machines in normal form; it is clear that a simple coding argument can show that ternary normal form Post machines can then be simulated by binary normal form Post machines.

Theorem 3 For every binary Post machine P we can construct an equivalent Post machine P' (with an extra alphabet symbol) in normal form.

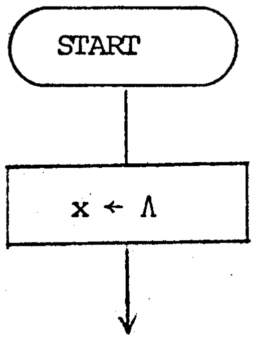
Proof P' simulates P , but keeps an end-marker symbol # on the end of x . We construct P' from P as follows:

Let $\boxed{\text{copy }(x)}$ denote the following

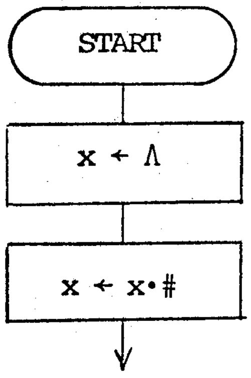


(This removes the end-marker from x .)

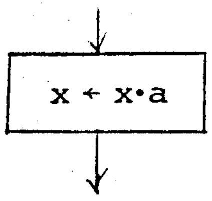
Then



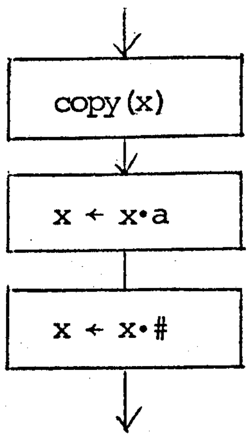
in P becomes



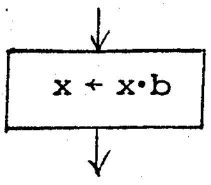
in P'



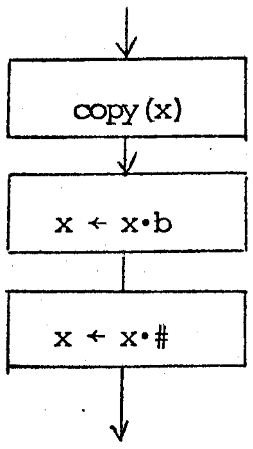
in P becomes



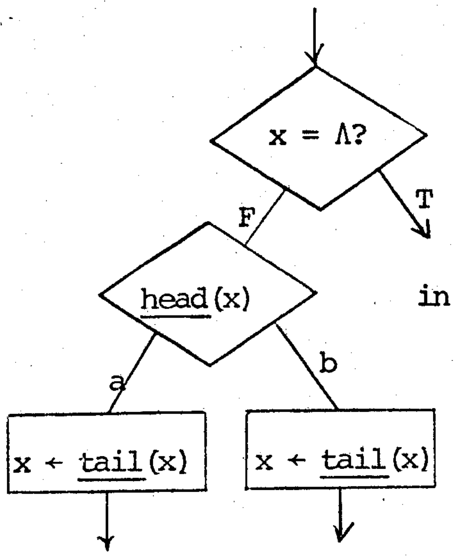
in P'



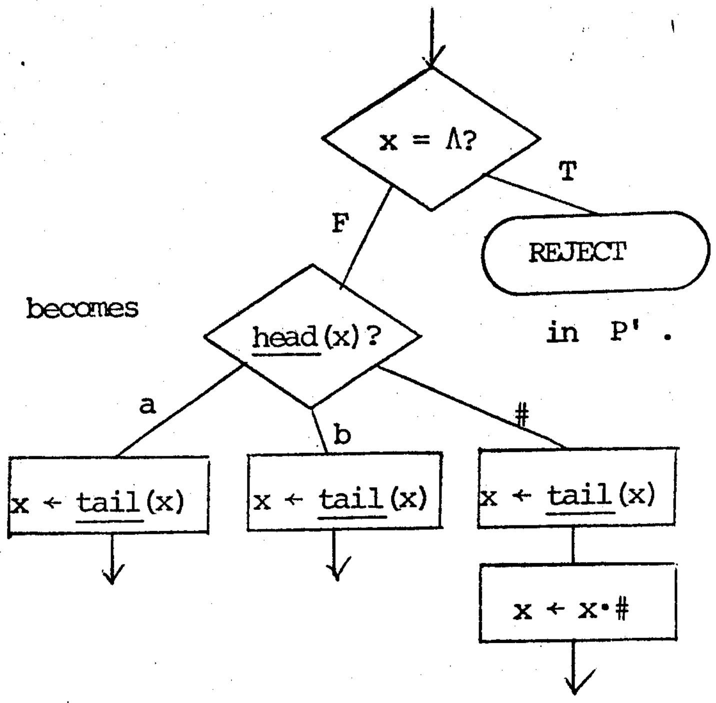
in P becomes



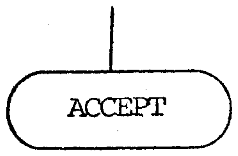
in P'



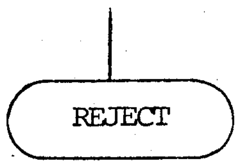
in P becomes



in P' .



and



remain the same in P' .

Consider the above fragments as elementary pieces of P and P' ; the construction ensures that the computations of P and P' follow the same path, and at each step (i.e. between fragments) if $x = \alpha$ in P then $x = \alpha\#$ in P' (and so x is never empty at the beginning of a reading statement fragment of P'). To see this, note that this is clearly true after the start statement and the relationship is preserved by all the other fragments. (The purpose of $\boxed{\text{copy}(x)}$ in P' is to convert $x = \alpha\#$ to $x = \alpha$.) Hence P' is equivalent to P i.e. it accepts/rejects if and only if P accepts/rejects. □