Math. Systems Theory 12, 205–211 (1979)

Mathematical
Systems Theory

A Difference in Expressive Power Between Flowcharts and Recursion Schemes

Nancy A. Lynch[1 2]

School of Information and Computer Science, Georgia Institute of Technology, Atlanta, Georgia

Edward K. Blum[1]

Department of Mathematics, University of Southern California, Los Angeles, California

Abstract. We show the existence of a single interpretation for which no flowchart produces the same results as a particular recursion scheme.

1. Introduction

In [1], Paterson and Hewitt construct a recursion scheme not "strongly equivalent" to any flowchart scheme. That is, any flowchart scheme will produce results different from those of their recursion scheme, for some interpretation of the basic function and predicate symbols. The choice of interpretation witnessing the difference, however, depends on the flowchart. In this paper, we prove a strengthened version of their result, for which the choice of interpretation is independent of the flowchart. In other words, we obtain a single interpretation for which no flowchart produces the same results as the recursion scheme of [1].

The recursion scheme under consideration has all of its basic functions either unary or 0-ary (constant). It is shown in [2] that for any recursion scheme with this restriction and for any interpretation "of sufficient power", there is a flowchart producing the same results for that interpretation. (The "sufficient power" requirement assures that flowcharts over the interpretation are capable

[1]Partially supported by NSF Grant DCR75-02373.
[2]Partially supported by NSF Grant MCS77-15628.

0025/5661/79/0012-0205$01.40

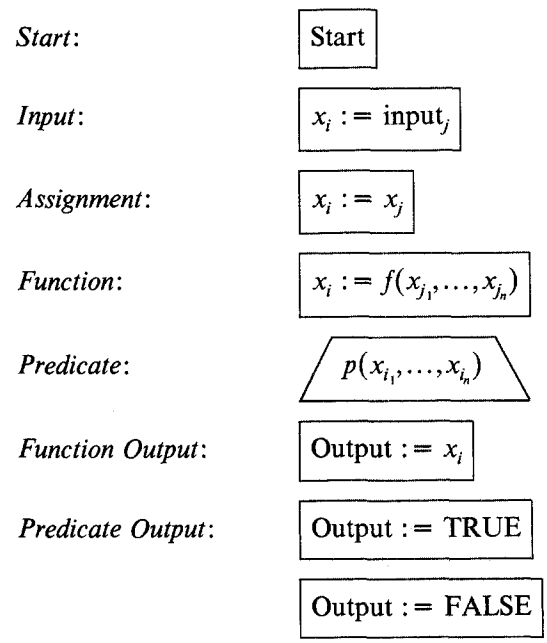
of simulating the partial recursive functions in some coding, and thereby are capable of simulating the control structure of recursion schemes.) Thus, the single interpretation we obtain will necessarily be of less than this power. We note that all of the interpretations produced in [1] are similarly of less than "sufficient power."

The construction we use originates in [1] but becomes more complicated, since we must patch together a single "diagonalizing interpretation" for all flowcharts.

2. Notation and Definitions

An *algebra* $\mathbb{S} = \langle D_{\mathbb{S}}; F_{\mathbb{S}}; P_{\mathbb{S}} \rangle$ is a set $D_{\mathbb{S}}$, the *domain of* $\mathbb{S}$, together with finite (indexed) sets $F_{\mathbb{S}}$ and $P_{\mathbb{S}}$ of partial functions (i.e. operations) and partial predicates (i.e. relations) on $D_{\mathbb{S}}$. Constants are 0-ary functions.

If $\mathbb{S}$ is an algebra, a *flowchart over* $\mathbb{S}$ is composed in the usual way from a finite number of boxes of the types:



where $f \in F_{\mathbb{S}}$ and $p \in P_{\mathbb{S}}$. A flowchart is either a *function flowchart*, in which case all output boxes are function output boxes, or a *predicate flowchart*, in which case all output boxes are predicate output boxes. Semantics are assumed to be obvious.

The flowcharts to be considered in this paper are all predicate flowcharts, and the algebras $\mathbb{S}$ all have $D_{\mathbb{S}} = \{0,1\}^*$, $F_{\mathbb{S}} = \{\lambda, 0\text{succ}, 1\text{succ}\}$, where λ is the empty string, $0\text{succ}(x) = x0$ and $1\text{succ}(x) = x1$. Moreover, $P_{\mathbb{S}} = \{p\}$, where p is a unary total recursive predicate.

We do not require a formal characterization of the class of recursion schemes, since any formulation broad enough to include the single recursion scheme we consider is sufficient.

If $\mathbb{S}$ is an algebra, then an equivalence relation, R, on $D_{\mathbb{S}}$ is a *congruence on* $\mathbb{S}$ if

(a) for every $f \in F_{\mathbb{S}}$, if $x_i R y_i$, $1 \leqslant i \leqslant n$, then either $f(x_1,\ldots,x_n)$ and $f(y_1,\ldots,y_n)$ are both undefined, or $f(x_1,\ldots,x_n) R f(y_1,\ldots,y_n)$,

and

(b) for every $p \in P_{\mathbb{S}}$, if $x_i R y_i$, $1 \leqslant i \leqslant n$, then either $p(x_1,\ldots,x_n)$ and $p(y_1,\ldots,y_n)$ are both undefined or $p(x_1,\ldots,x_n) = p(y_1,\ldots,y_n)$.

$R_{\max}(\mathbb{S})$, the *maximal congruence of* $\mathbb{S}$, is $\bigcup_{R \text{ a congruence on } \mathbb{S}} R$. It is straightforward to check that $R_{\max}(\mathbb{S})$ is a congruence on S.

Let N denote the nonnegative integers.

If $x,y \in \{0,1\}^*$, then y is said to be an *extension* of x if x is a prefix of y. If $x \in \{0,1\}^*$, $|x|$ denotes the length of x.

3. The Theorem

We begin with two technical lemmas.

Lemma 1. *Let* $\mathbb{S} = \langle \{0,1\}^*; \lambda, 0\text{succ}, 1\text{succ}; p\}$, *$p$ a total unary predicate. Let* $s \in \{0,1\}^*$, $g \geqslant 1$. *Assume p has the following properties*:

(a) *If x is an extension of s, then* $p(x) = \text{TRUE}$ *iff* $|x| = |s| + g$.

(b) *If x is not an extension of s and* $p(x) = \text{TRUE}$, *then* $|x| \leqslant |s|$.

Then $R_{\max}(\mathbb{S})$ *has at most* $2^{|s|} + g + 1$ *congruence classes.*

Proof. Consider equivalence relation R whose classes are

(a) $\{x\}$ for each x of length $\leqslant |s| - 1$,

(b) $\{x : x$ is a length k extension of $s\}$ for each $|s| \leqslant k \leqslant |s| + g - 1$,

(c) $\{x : x$ is an extension of s of length $|s| + g\} \cup$
$\{x : x \neq s$ and $|x| = |s|$ and $p(x) = \text{TRUE}\}$, and

(d) $\{x : x$ is an extension of s of length greater than $|s| + g\} \cup$
$\{x : x \neq s$ and $|x| = |s|$ and $p(x) = \text{FALSE}\} \cup$
$\{x : x$ is not an extension of s and $|x| > |s|\}$.

R is a congruence on $\mathbb{S}$, having $2^{|s|} + g + 1$ classes, so by definition, $R_{\max}$ can have no more classes. □

Lemma 2. *Let* $x,y,z \in \{0,1\}^*$, $k \in N$. *Then* $|\{w \in \{0,1\}^* : |w| = k$ *and some extension of w is in* $xy^*z\}| \leqslant k + 1$.

Proof. Straightforward. □

Theorem. *There exists an algebra* $\mathbb{S} = \langle \{0,1\}^*; \lambda, 0\text{succ}, 1\text{succ}; p\rangle$ *(p a unary total recursive predicate) and a partial unary predicate q such that q is computed by*

a recursion scheme interpreted over $\mathbb{S}$, but q is not computed by any flowchart over $\mathbb{S}$.

Proof. Let $(F_n)_{n\in N}$ be an enumeration of the unary predicate flowcharts with operation symbols in $\{\lambda, 0\text{succ}, 1\text{succ}, P\}$. (Here, we consider flowcharts which are identical up to renaming of variables to be the same.) We assume that F_n has at most n flowchart boxes and n locations. As [1], we consider the recursion scheme

$$Q(x) = \text{if } P(x) \text{ then TRUE else } Q(0\text{succ}(x)) \wedge Q(1\text{succ}(x)).$$

When interpreted over $\mathbb{S}$ above, with the symbol P interpreted as the predicate p, this scheme computes a (parametrized) partial predicate q_p. We assume semantics of recursion schemes are such that q_p is always either TRUE or undefined. Assuming p is total, q_p is TRUE for input x if and only if p is true for all nodes in some frontier of the tree of extensions of x.

We construct p in effective stages starting with stage 0 so that q_p satisfies the needed conditions on q in the statement of the theorem. The effect of stage n will be to insure that flowchart F_n does not compute q_p. Let p_0 denote the identically FALSE predicate. Let p_n, $n \geqslant 1$ denote the predicate defined by stages 0 through $n-1$. We will insure that $p_n(x)=\text{TRUE}$ implies $p_{n+1}(x)=\text{TRUE}$ for all n,x. The eventual p will be defined by $p(x)=\text{TRUE}$ iff $p_n(x)=\text{TRUE}$ for some n. To make p recursive, we construct (simultaneously with the construction of p) a recursive $s: N\to\{0,1\}^*$ such that

(a) $s(0)=\lambda$,

(b) for all n, $s(n)$ is a proper prefix of $s(n+1)$,

(c) if $p_n(x)=\text{TRUE}$, then $|x| \leqslant |s(n)|$ and $x \neq s(n)$,

and

(d) if $p_{n+1}(x)=\text{TRUE}$ and $p_n(x)=\text{FALSE}$, then x is a proper extension of $s(n)$.

The value of $s(n)$ will be defined before stage n.

Stage n. Choose any g so that $2^g \geqslant (2^{|s(n)|}+g+1)^n(n)+2$ and $2^{g+1} > (2^{|s(n)|}+g+1)^n(n^2)(2n+1+n(|s(n)|+g+2))$. Let p' be the total predicate defined by $p'(x)=$ TRUE iff either $p_n(x)=\text{TRUE}$ or $x=s(n)y$ for some y with $|y|=g$. There are two possibilities:

Case 1. F_n on input $s(n)$ with λ, 0succ and 1succ interpreted in the usual way and with P interpreted as p' halts (with either TRUE or FALSE as output) within $(2^{|s(n)|}+g+1)^n(n)$ steps.

Because of the bound on g, there exist y_n, z_n, $y_n \neq z_n$ with $|y_n|=|z_n|=g$ such that no extension of $s(n)y_n$ or of $s(n)z_n$ was generated in any location during the computation of F_n on $s(n)$ with p'. Let $p_{n+1}(x)=\text{TRUE}$ iff $p'(x)=\text{TRUE}$ and $x \notin \{s(n)y_n, s(n)z_n\}$. Let $s(n+1)=s(n)y_n$. Now F_n on $s(n)$ with p_{n+1} behaves exactly as F_n on $s(n)$ with p', as far as location contents, predicate answers and output are concerned. Moreover, since only extensions of $s(n)y_n$ will be added to the set of TRUE arguments at later stages, F_n on $s(n)$ with p will behave in exactly the same way, halting with output TRUE or FALSE. However, $q_p(s(n))$ is undefined since $p(x)$ is not TRUE for any extension x of $s(n)z_n$. Thus F_n cannot compute q_p.

Case 2. F_n on $s(n)$ with p' fails to halt within $(2^{|s(n)|}+g+1)^n(n)$ steps.

In this case, F_n on $s(n)$ with p' fails to halt at all. For there are at most $2^{|s(n)|}+g+1$ classes in the maximal congruence $R_{\max}(\langle\{0,1\}^*;\lambda,0\text{succ},1\text{succ};p'\rangle)$, by Lemma 1. Since there must be two times during the computation when all locations have congruent contents and control is at the same flowchart box, the computation is in a loop.

Claim. There is an extension s' of $s(n)$, $|s'|=|s(n)|+g+1$ such that no extension of s' is generated during the (non-halting) computation of F_n on $s(n)$ with p'.

Assuming this claim is true, we can let $s(n+1)=s'$, $p_{n+1}=p'$. Then F_n on $s(n)$ with p will not halt, but $q_p(s(n))=\text{TRUE}$, so F_n cannot compute q_p.

It remains to verify the claim by analyzing the set of values that can be generated by F_n on $s(n)$ with p'. Two of the first $(2^{|s(n)|}+g+1)^n(n)+1$ steps must take control to the same box in F_n with all corresponding location contents congruent (in $R_{\max}(\langle\{0,1\}^*;\lambda,0\text{succ},1\text{succ};p'\rangle)$). Thereafter, the computation is in a loop, repeating a sequence of flowchart boxes and n-tuples of congruence classes with period $\pi\leqslant(2^{|s(n)|}+g+1)^n(n)$. (We assume for simplicity that there are exactly n locations.)

We require notation for location contents. For $1\leqslant i\leqslant n$, $1\leqslant j$, $1\leqslant k\leqslant\pi$, let $c(i,j,k)$ denote the contents of location i after j entrances into the loop and k steps into that iteration of the loop. For consistency, let $c(i,0,\pi)$ denote the contents of location i at the first entrance to the loop. More precisely, $c(i,0,\pi)$ and $c(i,1,\pi)$ are congruent for all i, and the associated pair of steps of the computation is the first-completed pair of steps at which all corresponding location contents are congruent.

Different values of $c(i,j,k)$ are closely related. In particular, for every (i,k), there exist $w(i,k)\in\{0,1\}^*$ and $h(i,k)\in\{1,\ldots,n\}$ with either

(a) $c(i,j,k)=w(i,k)$ for all $j\geqslant 1$, or
(b) $c(i,j,k)=c(h(i,k),j-1,\pi)w(i,k)$ for all $j\geqslant 1$.

Intuitively, for fixed (i,k), either $c(i,j,k)$ is always built up in the same way from λ (so case (a) applies) or else $c(i,j,k)$ is always built up in the same way from the contents of the same location at the end of the last iteration (so case (b) applies). Case (b) may be further subdivided as follows:

(b1) There exists l, $1\leqslant l\leqslant n$, with

$$c(\underbrace{h(h(h(\ldots h(h}_{lh's}(i,k),\pi)\ldots),\pi),\pi),j,\pi)=$$

$$w(\underbrace{h(h(h(\ldots h(h}_{lh's}(i,k),\pi)\ldots),\pi),\pi),\pi)\text{ for all }j\geqslant 1.$$

Intuitively, this subcase occurs if when we trace a chain of dependencies backwards from (i,k) (seeing whether $c(h(i,k),j-1,\pi)$ is built up from λ or from the contents of a location at the end of the last iteration, and so on), that chain

will eventually terminate in a location whose value is built up from λ. l may be chosen to be at most n, since there are only n locations.

(b2) There exist $1 \leqslant l \leqslant m \leqslant n+1$ with

$$\underbrace{h(h(h(\ldots h(h(i,k),\pi)\ldots,\pi),\pi),\pi)}_{l\ h's} =$$

$$\underbrace{h(h(h(\ldots h(h(i,k),\pi)\ldots,\pi),\pi),\pi)}_{m\ h's}.$$

Intuitively, this subcase occurs if when we trace the above chain of dependencies backwards from (i,k), the dependent locations eventually repeat. Again, l and m may be chosen to be at most $n+1$, since there are only n locations.

Now we classify the values $c(i,j,k)$ produced for (i,k) satisfying each of (a), (b1) and (b2). For (i,k) satisfying (a), there is exactly one string $c(i,j,k)$ for $j \geqslant 1$. For (i,k) satisfying (b1), there are at most $n+1$ strings $c(i,j,k)$ for $j \geqslant 1$. (Sufficiently large j allow enough steps in the dependency chain for termination to occur.) For (i,k) satisfying (b2), there are strings $x_1,\ldots,x_n,y,z$ such that all $c(i,j,k)$, $j \geqslant n$, are in $\bigcup_{r=1}^{n} x_r y^* z$. (Sufficiently large j allow enough steps in the dependency chain for repeating of locations to occur.)

Now we can analyze the complete set of values that can be generated by F_n on $s(n)$ with p'. At most $(2^{|s(n)|}+g+1)^n(n^2)$ values are produced up to and including the first entrance to the loop. (i,k) satisfying (a) or (b1) cause a *total* of at most $(2^{|s(n)|}+g+1)^n(n^2)(n+1)$ values of $c(i,j,k)$, $j \geqslant 1$, to be produced. (i,k) satisfying (b2) cause a total of at most $(2^{|s(n)|}+g+1)^n(n^2)(n-1)$ values of $c(i,j,k)$ to be produced for $j<n$. And finally, (i,k) satisfying (b2) yield a total of at most $(2^{|s(n)|}+g+1)^n(n^2)(n)$ choices of x,y,z with arbitrarily many values of $c(i,j,k)$, $j \geqslant n$, in xy^*z. In summary, the values generated by F_n on $s(n)$ with p' are at most $(2^{|s(n)|}+g+1)^n(n^2)(2n+1)$ "sporadic" values, plus elements of at most $(2^{|s(n)|}+g+1)^n(n)$ sets of the form xy^*z. By Lemma 2, there are at most $(2^{|s(n)|}+g+1)^n(n^2)(2n+1+n(|s(n)|+g+2))$ strings of length $|s(n)|+g+1$ with extensions in this set of generated values. But since there are 2^{g+1} extensions of $s(n)$ of length $|s(n)|+g+1$, the bound on g suffices to justify the claim. □

4. Conclusions

In [2], the beginning of a theory of relative complexity (and computability) of algebras is presented. For definiteness, flowcharts and their runtimes are used as a programming language and complexity measure. The present theorem warns that the restriction to flowcharts is of some significance, even for computability. That is, for some algebras "computability by flowcharts" is not the same concept as "computability by recursive programs." For others, of course, they are the same.

One would expect similar distinctions among algebras to exist for complexity. Namely, over some algebras, flowcharts should be able to compute about as

efficiently as recursion schemes, whereas over others there should be a complexity difference. Different algebras have the power to simulate the control structure of recursion schemes with different overhead costs. Work remains to be done in establishing these distinctions.

Expressiveness results similar to the theorem of this paper and analogous complexity results should be obtainable for other commonly-studied scheme classes.

References

1. M. Paterson and C. Hewitt, Comparative Schematology, *MIT AI Lab Memo No. 201*, Nov., 1970.
2. N. Lynch and E. K. Blum, *Relative Complexity of Algebras*, Submitted for publication.

Received February 9, 1978 and in revised form July 24, 1978 and November 6, 1978 and in final form January 8, 1979