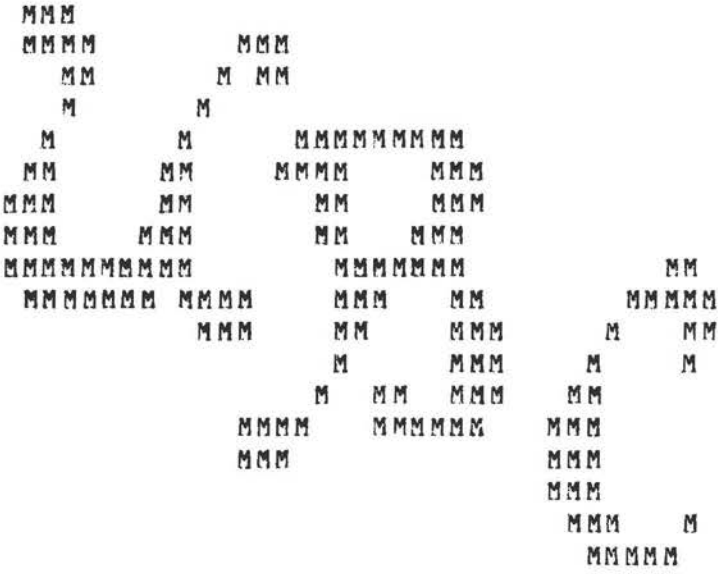


Coroutines in a Theory of Programmable Machines

by

John L. Baker

Technical Report 77-8

July 1977

Department of Computer Science
University of British Columbia
Vancouver, B. C.

Coroutines in a Theory of Programmable Machines *

John L. Baker

Department of Computer Science

University of British Columbia

ABSTRACT

It is shown that, in the author's theory of programmable machines, the composition of functions computable by programs is in some important cases computable by a program constructed to use the given programs as coroutines. To illustrate the utility of this result, a characterization of the full AFLs in terms of programmable machines is established with its help.

* Work supported in part by the National Research Council of Canada (Grant A7882).

0. Introduction.

Presentation of the theory of computation as a theory of programmable machines has the advantage of diminishing the distance between theoretical and practical work concerned with programming principles and their application. It seems clear that such presentation appeals directly to programming experience, so that mastery of its techinques and results by practical workers is not unduly impeded by irrelevant technicalities. Furthermore, such presentation affords theoretical workers direct access to the wisdom embodied in programming techniques.

This paper presents an extremely useful technical theorem concerning the composition of functions computable on programmable machines, together with a proof which, being based on the idea of coroutines, illustrates the application of programming experience in the development of theory. To illustrate the usefullness of the theorem itself, a characterization of the full AFLs in terms of programmable machines is here established with its help.

The following notation is used here: Card X denotes the cardinality of the set X. $X\backslash Y$ denotes the difference $\{x \in X \mid x \notin Y\}$. $\square$ denotes the empty set, 1 the set $\{0\}$ and 2 the set $\{0,1\}$ (when convenient), and $\mathfrak{N}$ the set $\{0,1,2\ldots\}$ of natural numbers.

Members of cartesian products are expressed as ordered tuples or as functions defined on an index set, according to convenience. However, it is assumed that cartesian products are defined by a universal construction, so that, for example, cartesian product as an operation on sets is associative and commutative.

$X+Y$ denotes the disjoint union of sets X and Y, similarly assumed to be defined by a universal construction. Members of $X+Y$ are usually expressed as if they were members of $X \cup Y$, with $X \cap Y = \square$.

"Function" here means "partial function". Specifically, $f:X\rightarrow Y$ means that f is a function (single-valued relation) defined for some elements of the set X and taking values in Y. $domf=\{x|y=f(x)$ for some $y\varepsilon Y\}$. $ranf=\{y|y=f(x)$ for some $x\varepsilon X\}$. As usualy, the barred arrow specifies a function by its action on an element. $f:x\mapsto y$ means (in the proper context) $y=f(x)$. If X is a set, Id_X denotes the identity relation (or function) on X.

$\circ$ denotes composition of (partial) functions, and is always defined. $z=[g\circ f](x)$ if and only if there is some y such that $y=f(x)$ and $z=g(y)$. Thus $dom(g\circ f)=\{x|f(x)\varepsilon domg\}$ and $ran(g\circ f)=\{g(y)|y\varepsilon ranf\}$.

If A is a set, then A^* denotes the set of strings over A as alphabet (the free monoid generated by A). $<>$ is the empty string (the identity of A^*), A^+ is the set of nonempty strings over A. For $x\varepsilon A^*$, $|x|$ is the <u>length</u> of x. ($|x|=0$ if and only if $x=<>$.)

1. Coroutines compute compositions.

The idea behind the above aphorism is presented in Knuth (1975) as "an important relation between coroutines and multiple-pass algorithms". The heart of it is this: Given two programs which compute functions by accepting input and producing output, one can construct a program of the following form to compute the composition of the functions: Invoke the first given program (as a subroutine), storing output as it is produced; then invoke the second given program, using the previously stored output as input. The resulting program implements a two-pass algorithm for the composition. If both the first-pass storing and second-pass retrieving are sequential, then there is an equivalent one-pass algorithm which can be implemented using the given programs as coroutines.

This idea can be presented quite explicitly in the theory of programmable machines outlined in Baker (1977). The theorem which embodies it (1.17 here) has considerable technical utility in that theory. In addition, that theorem and its proof indicate a potentially important and somewhat novel attitude toward the role of formal languages in the theory of computation.

For the most part, the theory of computation is considered to provide models for the external behavior of programs or systems. That is, it is applied to questions of the form what functions are computable (with given resources or techniques), and at what cost? -- it inquires into the nature of computing processes through their effects. With respect to the theory as a source for external models of computation, formal languages have been useful in two ways: The membership problem for languages in certain families has been a touchstone for comparison of computing resources; and some particularly important real-world computing activities, namely the translation or interpretation of programming and natural languages, have been guided in their development by theoretical models of lan-

guage processing.

If, on the other hand, it is considered suitable that the theory of computation provide models for internal structure of programs or systems--their decomposition into distinct computing processes or agents, in particular--, then formal languages may have a further significance: A particular decomposition of a program or system is somehow characterized by the language comprising the sequences of communications between its distinct parts which occur or can occur during computations. (Such a language is a formal language in the usual sense--a set of strings over a finite alphabet--because buffers and channels have finite capacity in practice, so that each single communication is chosen from a finite set.)

The quest for models of internal structure is supported by presentation of the theory of computing as a theory of programmable machines, particularly by the inclusion of a notion of product of devices (1.06 here), and the attitude toward formal languages just described is pretty clearly represented, in connection with that notion, in theorem 1.17 here.

The following fundamental definitions, constructions, and results (1.01 through 1.10) are included here for completeness. Consult Baker (1977) for examples, proofs, and motivational remarks concerning them.

1.01. A program Π comprises the following:

Π_Q , a finite set, the nodes;

$\Pi_S \in \Pi_Q$, the start node;

Π_A , a partial function with $\mathrm{dom}\Pi_A \subset \Pi_Q$, the action function;

Π_B , a partial function with $\mathrm{dom}\Pi_B \subset \mathrm{dom}\Pi_A \times U$ for some finite set U, and with $\mathrm{ran}\Pi_B \subset \Pi_Q$, the branching function.

It is also convenient to define

$\Pi_T = \Pi_Q \setminus \mathrm{dom}\Pi_A$, the terminal nodes;

$\Pi_C = \mathrm{ran}\Pi_A$, the commands;

$\Pi_V(\zeta) = \{i \mid \langle\zeta,i\rangle \in \mathrm{dom}\Pi_B\}$, the valence of ζ, defined for each $\zeta \in \Pi_Q$;

$\Pi_U = \cup\{\Pi_V(\zeta) \mid \zeta \in \Pi_Q\}$, the unified set of valences.

1.02. A device $\mathcal{D}$ comprises the following:

$\mathcal{D}_Q$, $\mathcal{D}_S$, $\mathcal{D}_T$, sets, the memory, input, and output sets;

$\mathcal{D}_I: \mathcal{D}_S \to \mathcal{D}_Q$, $\mathcal{D}_O: \mathcal{D}_Q \to \mathcal{D}_T$, partial functions, the input and output functions;

$\mathcal{D}_C$, a set, the commands;

$\mathcal{D}_G$, a partial function with $\mathrm{dom}\,\mathcal{D}_G \subset \mathcal{D}_C \times \mathcal{D}_Q$ and $\mathrm{ran}\,\mathcal{D}_G \subset \mathcal{D}_Q \times U$ for some set U, the general interpretation.

It is also convenient to define, for each $\alpha \in \mathcal{D}_C$,

$\mathcal{D}_V(\alpha) = \{i \mid \mathcal{D}_G(\alpha,m) = \langle m',i\rangle \text{ for some } m,m'\}$, the valence of α;

$\mathcal{D}_\alpha: \mathcal{D}_Q \to \mathcal{D}_Q \times \mathcal{D}_V(\alpha): m \mapsto \mathcal{D}_G(\alpha,m)$, the interpretation of α.

1.03. If Π is a program and $\mathcal{D}$ a device, then $\mathcal{C}(\Pi,\mathcal{D})$, the set of computations by Π on $\mathcal{D}$, is the set of sequences $\langle\zeta_0,m_0\rangle\ldots\langle\zeta_k,m_k\rangle$ in $\Pi_Q \times \mathcal{D}_Q$ in which, for all $j \in \{1,2\ldots,k\}$, $\mathcal{D}_{\Pi_A(\zeta_{j-1})}(m_{j-1}) = \langle m_j,i\rangle$ and $\Pi_B(\zeta_{j-1},i) = \zeta_j$ for some $i \in \Pi_V(\zeta_{j-1})$.

$\mathcal{C}_T(\Pi,\mathcal{D})$, the set of terminating computations by Π on $\mathcal{D}$, is the set of sequences $\langle\zeta_0,m_0\rangle\ldots\langle\zeta_k,m_k\rangle$ as above in which $\zeta_k \in \Pi_T$.

The length of a computation of the above form is k.

1.04. Lemma. If Π is a program, $\mathcal{D}$ a device, and $\langle\zeta_0,m_0\rangle \in \Pi_Q \times \mathcal{D}_Q$, then there is at most one sequence $\langle\zeta_0,m_0\rangle\ldots\langle\zeta_k,m_k\rangle$ in $\mathcal{C}_T(\Pi,\mathcal{D})$.

1.05. If Π is a program and $\mathcal{D}$ a device, then $\mathcal{D}_\Pi$, the function computed by Π on $\mathcal{D}$, is a partial function defined thus:

$\mathcal{D}_\Pi: \mathcal{D}_S \to \mathcal{D}_T : x \mapsto \mathcal{D}_O(m)$,

where $\langle \Pi_S, \mathcal{D}_I(x)\rangle \ldots \langle \zeta, m\rangle \in \mathcal{C}_T(\Pi, \mathcal{D})$.

1.06. If J is a set and for each $j \in J$ $\mathcal{D}_j$ is a device, then a device $\times_{j\in J} \mathcal{D}_j$ is defined thus (writing $\mathcal{P}$ for the product $\times_{j\in J} \mathcal{D}_j$):

$\mathcal{P}_Q = \times_{j\in J} (\mathcal{D}_j)_Q$, $\mathcal{P}_S = \times_{j\in J} (\mathcal{D}_j)_S$, $\mathcal{P}_T = \times_{j\in J} (\mathcal{D}_j)_T$.

$\mathcal{P}_I : x \mapsto m$, where $m(j) = (\mathcal{D}_j)_I(x(j))$ for all $j \in J$.

$\mathcal{P}_O : m \mapsto y$, where $y(j) = (\mathcal{D}_j)_O(m(j))$ for all $j \in J$.

$\mathcal{P}_C$ is the disjoint union $+_{j\in J} (\mathcal{D}_j)_C$.

For each $\alpha \in \mathcal{P}_C$, if $\alpha \in (\mathcal{D}_j)_C$, then $\mathcal{P}_\alpha : m \mapsto \langle m', i\rangle$, where $m' = m$ except that $m'(j)$ is such that $(\mathcal{D}_j)_\alpha(m(j)) = \langle m'(j), i\rangle$.

1.07. If $\mathcal{D}$ and $\mathcal{E}$ are devices, then $\mathcal{D} < \mathcal{E}$ ($\mathcal{D}$ is reducible to $\mathcal{E}$) if and only if, whenever Π is a program, there is a program Π' such that $\mathcal{E}_{\Pi'} = \mathcal{D}_\Pi$.

$\mathcal{D} \sim \mathcal{E}$ ($\mathcal{D}$ is equivalent to $\mathcal{E}$) if and only if $\mathcal{D} < \mathcal{E}$ and $\mathcal{E} < \mathcal{D}$ as above.

1.08. If Π is a program and $\mathcal{D}$ a device, then Π is for $\mathcal{D}$ if and only if $\Pi_C \subset \mathcal{D}_C$ and $\Pi_V(\zeta) \subset \mathcal{D}_V(\Pi_A(\zeta))$ for all $\zeta \in \Pi_Q$.

1.09. Lemma. If Π is a program and $\mathcal{D}$ a device with $\mathcal{D}_C \neq \Box$, then there is a program Π' such that $\Pi'_\iota = \Pi_\iota$ for $\iota \in \{Q,S,T\}$, and $\mathcal{C}(\Pi', \mathcal{D}) = \mathcal{C}(\Pi, \mathcal{D})$.

1.10. Theorem. If G is a set of programs and $\mathcal{D}$ a device, then $\mathcal{D} \sim \mathcal{D}'$, where $\mathcal{D}'_\iota = \mathcal{D}_\iota$ for $\iota \in \{Q,S,T,I,O\}$, $\mathcal{D}'_C = \mathcal{D}_C + G$, $\mathcal{D}'_\alpha = \mathcal{D}_\alpha$ if $\alpha \in \mathcal{D}_C$, and, for each $\Gamma \in G$,

$\mathcal{D}'_\Gamma(m) = \langle m', \zeta\rangle \Leftrightarrow \langle \Gamma_S, m\rangle \ldots \langle \zeta, m'\rangle \in \mathcal{C}_T(\Gamma, \mathcal{D})$.

1.11. Notation. If G, $\mathcal{D}$, and $\mathcal{D}'$ are as in (1.10), we call the elements of G programmable operations for $\mathcal{D}$. We may define programs under that name

and use them as in programs for $\mathcal{D}'$ without further apology.

For any device $\mathcal{D}$, No is the programmable operation specified by $\underset{\cdot}{0}:$.

The following development (through 1.14) formalizes the observation that finite data structures are indistinguishable from control structures.

1.12. A device $\mathcal{D}$ has trivial input [resp. output] if and only if card $\mathcal{D}_S=1$ [resp. card $\mathcal{D}_T=1$] and dom $\mathcal{D}_I=\mathcal{D}_S$ [resp. dom $\mathcal{D}_O=\mathcal{D}_Q$].

A device $\mathcal{D}$ is trivial if and only if it has trivial input and output and $\mathcal{D}_Q$ is finite.

1.13. Theorem. If $\mathcal{D}$ is a device, $\mathcal{E}$ is a trivial device, and Π is a program, then there is a program Π' such that $\mathcal{D}_{\Pi'}(x)=y$ if and only if $\mathcal{D}\times\mathcal{E}_{\Pi}(x,s)=\langle y,t\rangle$, where $\mathcal{E}_S=\{s\}$ and $\mathcal{E}_T=\{t\}$. ($\mathcal{D}\times\mathcal{E}<\mathcal{D}$ if we take the point of view that $(\mathcal{D}\times\mathcal{E})_S=\mathcal{D}_S$ and $(\mathcal{D}\times\mathcal{E})_T=\mathcal{D}_T$.)

Proof: Define Π' by $\Pi'_Q=\Pi_Q\times\mathcal{E}_Q$, $\Pi'_S=\langle\Pi_S,\mathcal{E}_I(s)\rangle$,

$$\Pi'_A:\langle\zeta,u\rangle\mapsto\left\{\begin{array}{ll}\Pi_A(\zeta) & \text{if } \Pi_A(\zeta)\notin\mathcal{E}_C\\ \text{No} & \text{if } \Pi_A(\zeta)\in\mathcal{E}_C\end{array}\right\},$$

$$\Pi'_B:\langle\langle\zeta,u\rangle,i\rangle\mapsto\left\{\begin{array}{ll}\langle\Pi_B(\zeta,i),u\rangle & \text{if } \Pi_A(\zeta)\notin\mathcal{E}_C\\ \langle\Pi_B(\zeta,i'),u'\rangle & \text{if } i=0 \text{ and } \mathcal{E}_{\Pi_A(\zeta)}(u)=\langle u',i'\rangle\\ \text{undefined otherwise.}\end{array}\right.$$

Π' is as required, since clearly $\langle\zeta_0,\langle m_0,u_0\rangle\rangle\ldots\langle\zeta_k,\langle m_k,u_k\rangle\rangle\in\mathcal{C}(\Pi,\mathcal{D}\times\mathcal{E})$ if and only if $\langle\langle\zeta_0,u_0\rangle,m_0\rangle\ldots\langle\langle\zeta_k,u_k\rangle,m_k\rangle\in\mathcal{C}(\Pi',\mathcal{D})$.

1.14. Notation. By virtue of (1.13), we may include use of variables over finite sets in any program. Where appropriate, we specify initial values for such variables. The commands to be employed are:

$x \leftarrow a$, $x \leftarrow y$: assignment of constant or current variable values:

valence = {0};

$x=$: test of current value: valence is the set over which x varies.

These notions are to be understood in their usual programming language senses, assuming static storage allocation for variables.

We come now to the principal subject of this paper, the representation in our theory of the use of coroutines to compute the composition of functions. The following two devices are used to represent repectively the storage and retrieval of intermediate results in a two-pass computation. Their absence in (1.17(i)) represents the incorporation of intermediate results into the coroutine mechanism underlying an equivalent one-pass computation as described at the beginning of this section.

1.15. If A is a finite set, the output device $\mathrm{Out}^{(A)}$ is specified thus (omitting the superscript):

$\mathrm{Out}_Q = \mathrm{Out}_T = A^*$, $\mathrm{Out}_S = 1$.

$\mathrm{Out}_I : 0 \mapsto \langle\rangle$, $\mathrm{Out}_O = \mathrm{Id}_{A^*}$.

$\mathrm{Out}_C = \{0 \leftarrow a \mid a \in A\}$.

$\mathrm{Out}_{0 \leftarrow a} : x \mapsto \langle xa, 0 \rangle$.

1.16. If A is a finite set, the one-way input device $\mathrm{In1}^{(A)}$ is specified thus (omitting the superscript):

$\mathrm{In1}_Q = (A + \{\dashv\})^*$, $\mathrm{In1}_S = A^*$, $\mathrm{In1}_T = 1$.

$\mathrm{In1}_I : x \mapsto x\dashv$, $\mathrm{In1}_O : \langle\rangle \mapsto 0$ (undefined on nonempty strings).

$\mathrm{In1}_C = \{\delta\}$.

$\mathrm{In1}_\delta : ax \mapsto \langle x, a \rangle$ for all $a \in A + \{\dashv\}$ (undefined on $\langle\rangle$).

1.17. Theorem. If A and B are finite sets with A⊂B, $\mathfrak{D}$ and $\mathcal{E}$ are devices, and Φ and Ψ are programs, then there is a program Π such that the following are equivalent:

(i) $\mathfrak{D}\times\mathcal{E}_{\Pi}(x_1,x_2)=\langle y_1,y_2\rangle$

(ii) $\mathfrak{D}\times\mathrm{Out}^{(A)}_{\Phi}(x_1,0)=\langle y_1,z\rangle$ and $\mathrm{Inl}^{(B)}\times\mathcal{E}_{\Psi}(z,x_2)=\langle 0,y_2\rangle$ for some $z\in A^*$.

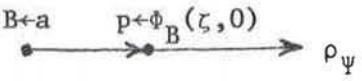
Proof: Let $P=(\{\Phi_S\}\cup\{\Phi_B(\zeta,0)\,|\,\Phi_A(\zeta)=0\leftarrow a \text{ for some } a\in A\})+\{T\}$, $Q=\{\Psi_S\}\cup\{\Psi_B(\zeta,a)\,|\,\Psi_A(\zeta)=\delta \text{ and } a\in A+\{\dashv\}\}$. By (1.09), assume Φ is for $\mathfrak{D}\times\mathrm{Out}^{(A)}$ and Ψ is for $\mathrm{Inl}^{(B)}\times\mathcal{E}$. Define Π with $\Phi_Q+\Psi_Q\subset\Pi_Q$, and using variables $\langle B,p,q\rangle$ over $\langle A+\{\dashv,-\},P,Q\rangle$ with initial values $\langle -,\Phi_S,\Psi_S\rangle$, thus:

$\Pi_S=\Phi_S$.

If $\Phi_A(\zeta)\in\mathfrak{D}_C$, then $\Pi_A(\zeta)=\Phi_A(\zeta)$, $\Pi_V(\zeta)=\Phi_V(\zeta)$, and $\Pi_B(\zeta,i)=\Phi_B(\zeta,i)$ for each $i\in\Phi_V(\zeta)$.

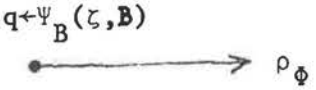
If $\Psi_A(\zeta)\in\mathcal{E}_C$, then $\Pi_A(\zeta)=\Psi_A(\zeta)$, $\Pi_V(\zeta)=\Psi_V(\zeta)$, and $\Pi_B(\zeta,i)=\Psi_B(\zeta,i)$ for all $i\in\Psi_V(\zeta)$.

If $\Phi_A(\zeta)=0\leftarrow a$, then Π includes

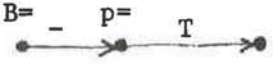


If $\zeta\in\Phi_T$, then Π includes

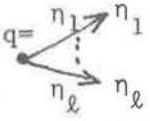
If $\Psi_A(\zeta)=\delta$, then Π includes

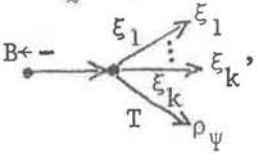


If $\zeta\in\Psi_T$, then Π includes



In addition, Π includes

ρ_Ψ:  , where $Q=\{\eta_1,\ldots,\eta_\ell\}$; and

ρ_Φ:  , where $\{\Phi_B(\zeta,0)\,|\,\Phi_A(\zeta)=0\leftarrow a \text{ for some } a\varepsilon A\}=\{\xi_1,\ldots,\xi_k\}$.

As suggested in the motivational remarks already given, Π is constructed to use Φ and Ψ as, respectively, producer and consumer coroutines. Φ is run until it is about to produce a symbol by executing an "$0\leftarrow a$" command. Control is then passed to Ψ, which is run until it is about to consume a symbol by a "δ" command. At that time, control is returned to Φ at the node it would have reached if the "$0\leftarrow a$" command mentioned had been executed. At the same time, linkage is set up to resume execution of Ψ at the node reached by following the "a" branch from that bearing the "δ" command mentioned. Alternate execution of Φ and Ψ continues in this manner until Φ halts. Ψ is then resumed as usual, but with linkage so arranged that, when it next reaches a "δ" node, it is left in control at the node reached by following the "$\dashv$" branch. Π then halts whenever Ψ does, provided does not reach any further "δ" nodes.

If the reader sees clearly how the construction given for Π is as just described, he need read no further in this section. Otherwise, he may proceed deductively, reading the further detail now to be given, or inductively, examining the example (1.18) below.

In the rest of this proof, denote $\mathcal{C}(\Phi,\mathcal{D}\times \mathrm{Out}^{(A)})$ by $\mathcal{C}(\Phi)$, $\mathcal{C}(\Psi,\mathrm{In}1^{(B)}\times\mathcal{E})$ by $\mathcal{C}(\Psi)$, and $\mathcal{C}(\Pi,\mathcal{D}\times\mathcal{E})$ by $\mathcal{C}(\Pi)$.

We first prove (i)$\Rightarrow$(ii). It is not difficult to prove, by induction on the length of the Π-computation about to be mentioned, that if $\zeta\varepsilon\Phi_Q{+}\Psi_Q$ and

$\langle \Pi_S, \hat{m}, \hat{n}, \langle -, \Phi_S, \Psi_S \rangle \rangle \ldots \langle \zeta, m, n, \langle a, \xi, \eta \rangle \rangle \in \mathcal{C}(\Pi)$ then there is some string $z \in A^*$ such that either $\zeta \in \Phi_Q$, $a = -$, $\langle \Phi_S, \hat{m} \rangle \rangle \ldots \langle \xi, m', z \rangle \ldots \langle \zeta, m, z \rangle \in \mathcal{C}(\Phi)$ for some m',

and $\langle \Psi_S, z, \hat{n} \rangle \ldots \langle \eta, \langle\rangle, n \rangle \in \mathcal{C}(\Psi)$

or $\zeta \in \Psi_Q$,

(either $\langle \Phi_S, \hat{m}, \langle\rangle \rangle \ldots \langle \xi, m, za \rangle \in \mathcal{C}(\Phi)$

or $\xi = T$ and $\langle \Phi_S, \hat{m}, \langle\rangle \rangle \ldots \langle \Theta, m, z \rangle \in \mathcal{C}(\Phi)$ for some $\Theta \in \Phi_T$), and

(either $\langle \Psi_S, za, \hat{n} \rangle \ldots \langle \eta, a, n' \rangle \ldots \langle \zeta, a, n \rangle \in \mathcal{C}(\Psi)$ for some n'

or $a = -$ and $\langle \Psi_S, z\dashv, \hat{n} \rangle \ldots \langle \eta, \langle\rangle, n' \rangle \ldots \langle \zeta, \langle\rangle, n \rangle \in \mathcal{C}(\Psi)$ for some n').

Now suppose (i). By construction of Π, $\langle \Pi_S, \mathcal{D}_I(x_1), \mathcal{E}_I(x_2), \langle -, \Phi_S, \Psi_S \rangle \rangle \ldots \langle \zeta, m, n \langle -, T, \eta \rangle \rangle \in \mathcal{C}(\Pi)$ for some $\zeta \in \Psi_T$, $\eta \in \Psi_Q$, $m \in \mathcal{D}_Q$, and $n \in \mathcal{E}_Q$, with $y_1 = \mathcal{D}_0(m)$ and $y_2 = \mathcal{E}_0(n)$. By the assertion above, there is some $z \in A^*$ such that $\langle \Phi_S, \mathcal{D}_I(x_1), \langle\rangle \rangle \ldots \langle \Theta, m, z \rangle \in \mathcal{C}(\Phi)$ for some $\Theta \in \Phi_T$ and $\langle \Psi_S, z\dashv, \mathcal{E}_I(x_2) \rangle \ldots \langle \zeta, \langle\rangle, n \rangle \in \mathcal{C}(\Psi)$. (ii) follows.

We next prove (ii)$\Rightarrow$(i). For $\xi \in P$, $\eta \in Q$, it is clear from the construction of Π that

(1) If $\langle \xi, m, \langle\rangle \rangle \ldots \langle \Theta, \tilde{m}, \langle\rangle \rangle \ldots \langle \zeta, m', a \rangle \in \mathcal{C}(\Phi)$, $\Phi_A(\Theta) = 0 \leftarrow a$,

$\langle \eta, \langle\rangle, n \rangle \ldots \langle \lambda, \langle\rangle, n' \rangle \in \mathcal{C}(\Psi)$, $\Psi_A(\lambda) = \delta$, and $\langle \lambda, a \rangle \in \mathrm{dom}\Psi_B$,

then $\langle \xi, m, n, \langle -, \xi, \eta \rangle \rangle \ldots \langle \zeta, m', n', \langle -, \Phi_B(\Theta, 0), \Psi_B(\lambda, a) \rangle \rangle \in \mathcal{C}(\Pi)$; and

(2) If $\langle \xi, m, \langle\rangle \rangle \ldots \langle \Theta, m', \langle\rangle \rangle \in \mathcal{C}(\Phi)$, $\Theta \in \Phi_T$,

$\langle \eta, \dashv, n \rangle \ldots \langle \lambda, \dashv, \tilde{n} \rangle \ldots \langle \zeta, \langle\rangle, n' \rangle \in \mathcal{C}(\Psi)$, $\Psi_A(\lambda) = \delta$, and $\langle \lambda, \dashv \rangle \in \mathrm{dom}\Psi_B$,

then $\langle \xi, m, n, \langle -, \xi, \eta \rangle \rangle \ldots \langle \zeta, m', n', \langle -, T, \Psi_B(\lambda, \dashv) \rangle \rangle \in \mathcal{C}(\Pi)$.

From (1,2) it follows by induction on $|z|$ that, for $\xi \in P$, $\eta \in Q$, $z \in A^*$,

(3) If $\langle \xi, m, \langle\rangle \rangle \ldots \langle \Theta, m', z \rangle \in \mathcal{C}(\Phi)$, $\Theta \in \Phi_T$, and

$\langle \eta, z\dashv, n \rangle \ldots \langle \zeta, \langle\rangle, n' \rangle \in \mathcal{C}(\Psi)$,

then $\langle \xi, m, n, \langle -, \xi, \eta \rangle \rangle \ldots \langle \zeta, m', n', \langle -, T, \eta' \rangle \rangle \in \mathcal{C}(\Pi)$ for some η'.

(ii)⇒(i) follows easily from (3).

1.18. Example. If Σ is a finite or countable set, the pushdown store device $Pds^{(\Sigma)}$ is specified thus (omitting the superscript):

$Pds_Q=\Sigma^*$, $Pds_S=Pds_T=1$.

$Pds_I: 0\mapsto<>$, $Pds_O: <>\mapsto 0$ (undefined on nonempty strings).

$Pds_C=\{\leftarrow P\}\cup\{P\leftarrow a \mid a\in\Sigma\}$.

$Pds_{\leftarrow p}: ax\mapsto<x,a>$ for all $a\in\Sigma$ (undefined on <>).

$Pds_{P\leftarrow a}: x\mapsto<ax,0>$.

Let Φ and Ψ be respectively the programs of Figure 1(a) and (b). Clearly $In1^{(\{a,b\})}\times Out^{(\{a,b\})}_{\Phi}: <x,0>\mapsto<0,f^{-1}(x)>$, where f is the homomorphism $a\mapsto a$, $b\mapsto bb$; and $dom(In1^{(\{a,b\})}\times Pds^{(\{a\})}_{\Psi})=\{a^ib^i \mid i\geq 0\}\times\{0\}$. (1.17) asserts that there is a program Π with $dom(In1^{(\{a,b\})}\times Pds^{(\{a\})}_{\Pi})=\{a^ib^{2i} \mid i\geq 0\}\times\{0\}$, and the proof of (1.17) gives a construction for such a program Π, as shown in figure 1(c). To get a clearer idea of how Π works, we can eliminate use of variables, using the construction of (1.13), then elide the resulting occurrences of the command "No" in an obvious way. The result is shown in figure 1(d).

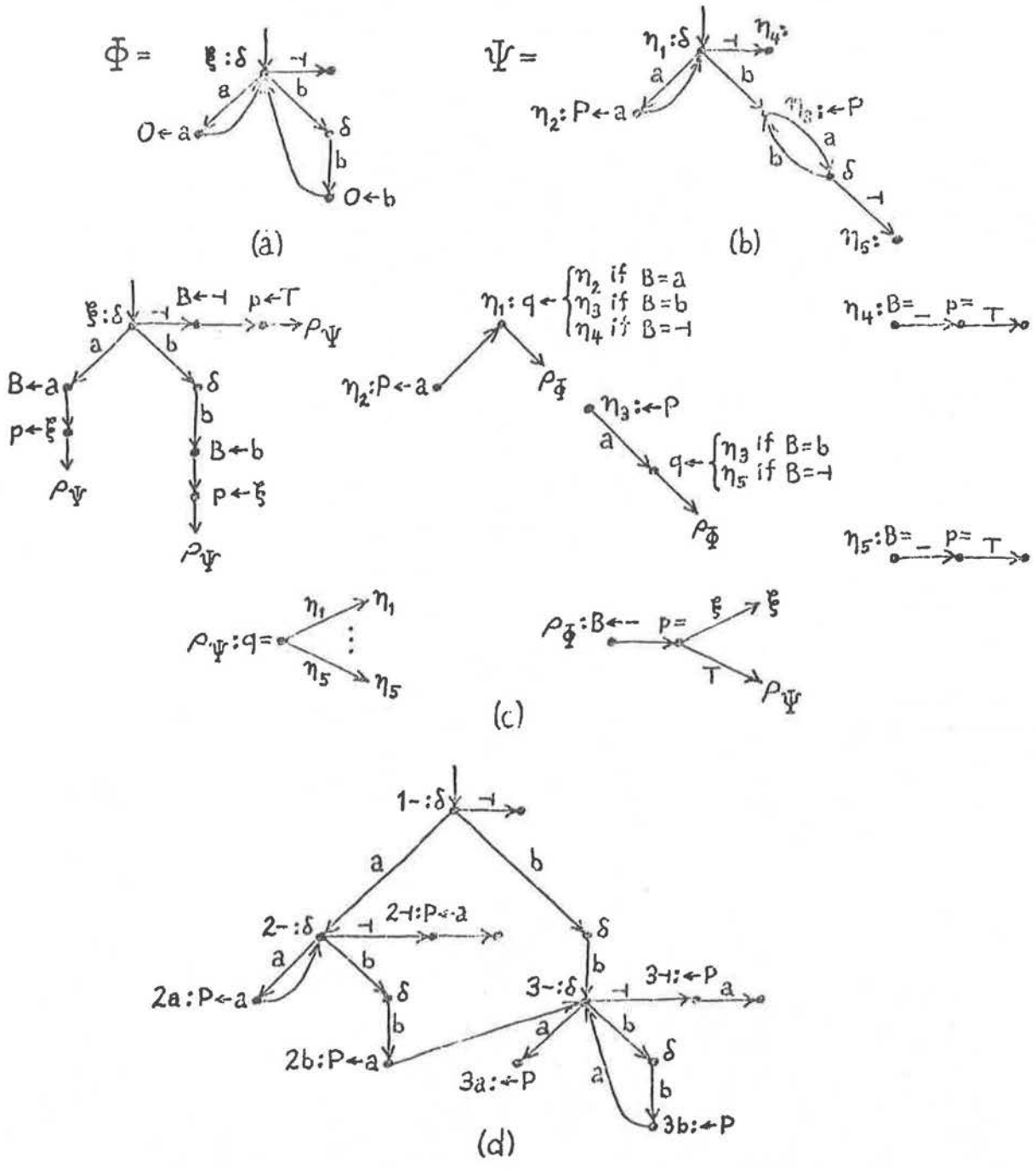


Figure 1. Programs of example 1.18, illustrating the coroutine construction in the proof of theorem 1.17. In (d), for i=1,2,3,

i- denotes $\langle \xi, \langle -, \xi, \eta_i \rangle \rangle$

ia denotes $\langle \eta_i, \langle a, \xi, \eta_i \rangle \rangle$

ib denotes $\langle \eta_i, \langle b, \xi, \eta_i \rangle \rangle$

i⊣ denotes $\langle \eta_i, \langle \dashv, T, \eta_i \rangle \rangle$.

2. A characterization of the full AFLs.

Theorem 1.17 here is technical in the sense that one cannot draw from it alone any important conclusions about the feasibility or cost of real-world computations. Although, as suggested above, it may have some philosophical importance with respect to the role of formal languages in the theory of computing, it is offered primarily in the hope that its use may allow some more directly applicable parts of the theory to be developed simply and clearly. To illustrate what can be done along these lines, here is a brief but nearly complete development of a characterization of the full AFLs in terms of programmable machines.

A full AFL is defined to be a family of languages closed under the operations union, concatenation, concatenation closure, intersection with regular sets, homomorphism, and inverse homomorphism. (See Ginsburg (1975).) We will see that a family $\mathcal{L}$ is a full AFL if and only if there is a machine $\mathcal{M}$ which is a programmable nondeterministic acceptor for $\mathcal{L}$ in the sense that $L \in \mathcal{L}$ if and only if there is a program Π such that Π running on $\mathcal{M}$ is a nondeterministic acceptor for L.

Nondeterminism, which plays an essential role in the characterization we are seeking, comes into the theory outlined in section 1 as a matter of interpretation. Specifically, a relation (nondeterministically) computed by a program Π on a device $\mathcal{D}$ will be defined as the projection of the function $\mathcal{D}_\Pi$ obtained by omitting a specified factor of $\mathcal{D}_S$ (presumed to be a product), the omitted component being thereby regarded as an auxiliary input selecting a particular path in the tree of computations by Π on $\mathcal{D}$ determined by the components not omitted. We will use the following as a standard auxiliary input device in this sense.

2.01. The auxiliary input device Aux is specified thus:

$Aux_Q = Aux_S = 2^*$, $Aux_T = 1$.

$Aux_I = Id_{2^*}$, $Aux_O : \langle\rangle \mapsto 0$ (undefined on nonempty strings).

$Aux_C = \{A\}$.

$Aux_A : ix \mapsto \langle x,i\rangle$ for $i \in 2$ (undefined on $\langle\rangle$).

2.02. Remark. For any set X, let $\mathfrak{R}_X$ denote the regular languages over X (as alphabet). It is easy to show, using familiar techniques, that $\{\mathrm{dom}(\mathrm{In1}^{(A)}{}_\Pi) \mid \Pi \text{ is a program}\} = \mathfrak{R}_A$ for any finite set A. Programs for $\mathrm{In1}^{(A)}$ are in this sense deterministic finite-state acceptors. Programs for $\mathrm{Aux}\times\mathrm{In1}^{(A)}$ are likewise nondeterministic finite-state acceptors in the sense that $\{x \mid \langle w,x\rangle \in \mathrm{dom}(\mathrm{Aux}\times\mathrm{In1}^{(A)}{}_\Pi)$ for some program Π and some $w \in 2^*\}$ is also $\mathfrak{R}_A$. Similarly, $\mathrm{Aux}\times\mathrm{Out}^{(A)}$ is a programmable nondeterministic generator, and we have $\{x \mid \langle 0,x\rangle \in \mathrm{ran}(\mathrm{Aux}\times\mathrm{Out}^{(A)}{}_\Pi)$ for some program $\Pi\} = \mathfrak{R}_A$.

The above uses of Aux are typical. In specifying relations defined computationally, we will always project away from Aux. Likewise, we will always project away from components with trivial input or output. In this connection, the following nomenclature is useful.

2.03. A machine $\mathcal{M}$ comprises the following:

$\mathcal{M}_\mathcal{D}$ an indexing function with $\mathcal{M}_\mathcal{D}(j)$ a device for all $j \in \mathrm{dom}\ \mathcal{M}_\mathcal{D}$;

$\mathcal{M}_K$, $\mathcal{M}_L \subset \mathrm{dom}\ \mathcal{M}_\mathcal{D}$, the indices active for input and output, respectively.

It is also convenient to define

$\mathcal{M}_J = \mathrm{dom}\ \mathcal{M}_\mathcal{D}$, the index set;

$\mathcal{M}_D = \underset{j \in \mathcal{M}_J}{\times} \mathcal{M}_\mathcal{D}(j) = \times(\mathrm{ran}\ \mathcal{M}_\mathcal{D})$, the underlying device;

$\mathcal{M}_S = \underset{j \in \mathcal{M}_K}{\times} (\mathcal{M}_\mathcal{D}(j))_S$, the input set; and

$\mathcal{M}_T = \underset{j \in \mathcal{M}_L}{\times} (\mathcal{M}_\mathcal{D}(j))_T$, the output set.

2.04. If $\mathcal{M}$ is a machine and Π a program, then the relation computed by Π on $\mathcal{M}$ is

$$\mathcal{R}(\Pi,\mathcal{M})=\{\langle x,y\rangle \in \mathcal{M}_S \times \mathcal{M}_T \mid (\mathcal{M}_D)_\Pi(\overline{x})=\overline{y} \text{ for some } \overline{x}\in(\mathcal{M}_D)_S \text{ and } \overline{y}\in(\mathcal{M}_D)_T \text{ with } \overline{x}(j)=x(j) \text{ for all } j\in\mathcal{M}_K \text{ and } \overline{y}(j)=y(j) \text{ for all } j\in\mathcal{M}_L\}$$

2.05. Notation. A machine $\mathcal{M}$ is ordinarily to be specified by exhibiting $\mathcal{M}_D$, thus specifying the index set $\mathcal{M}_J$ and the function $\mathcal{M}_\mathcal{D}$ implicitly. The sets $\mathcal{M}_K$ and $\mathcal{M}_L$ may also be specified implicitly by referring to devices which are factors in the product $\mathcal{M}_D$ as active for input or output.

In this paper, the devices $\text{In1}^{(A)}$ are active for input in all machines, the devices $\text{Out}^{(A)}$ are active for output in all machines, and none of $\text{In1}^{(A)}$, $\text{Out}^{(A)}$, Aux, or $\text{Pds}^{(\Sigma)}$ (example 1.18) is otherwise active for input or output. In particular, Aux is not active for input or output in any machine.

If $\mathcal{M}$ is a machine and $\mathcal{D}$ a device, then $\mathcal{M}\times\mathcal{D}$ denotes $\mathcal{M}_D\times\mathcal{D}$, which notation serves as in the first paragraph above to denote both a machine and a device.

The following is also convenient.

2.06. If $\mathcal{M}$ is a machine, then

$X\subset\mathcal{M}_S$ is $\mathcal{M}$-acceptable if and only if there is a program Π such that $X=\text{dom}\,\mathcal{R}(\Pi,\mathcal{M})$.

$X\subset\mathcal{M}_T$ is $\mathcal{M}$-generable if and only if there is a program Π such that $X=\text{ran}\,\mathcal{R}(\Pi,\mathcal{M})$.

In terms of the above nomenclature, the remark (2.02) asserts the equivalence of the following propositions, where A is a finite set:

(i) $X\in\mathfrak{R}_A$

(ii) X is $\text{In1}^{(A)}$-acceptable.

(iii) X is $\mathrm{Aux}\times\mathrm{Inl}^{(A)}$-acceptable.

(iv) X is $\mathrm{Aux}\times\mathrm{Out}^{(A)}$-generable.

We now proceed to develop a few basic facts about machines and the role of Aux (nondeterminism) in general, beginning with an obvious generalization of definition 1.07.

2.07. If $\mathcal{M}$ and $\mathcal{N}$ are machines, then $\mathcal{M} <_m \mathcal{N}$ ($\mathcal{M}$ is reducible (as a machine) to $\mathcal{N}$) if and only if, whenever Π is a program, there is a program Π' such that $\mathcal{R}(\Pi', \mathcal{N}) = \mathcal{R}(\Pi, \mathcal{M})$.

$\mathcal{M} \sim_m \mathcal{N}$ ($\mathcal{M}$ is equivalent (as a machine) to $\mathcal{N}$) if and only if $\mathcal{M} <_m \mathcal{N}$ and $\mathcal{N} <_m \mathcal{M}$.

2.08. Lemma. If $\mathcal{M}$ is a machine and $\mathcal{D}$ is a device such that $\mathcal{D}_O(\mathcal{D}_I(s))$ is defined for some $s \in \mathcal{D}_S$, then $\mathcal{M} <_m \mathcal{M}\times\mathcal{D}$, where $\mathcal{D}$ is not active for input or output in this product.

Proof: Whenever Π is a program for $\mathcal{M}_D$, $(\mathcal{M}_D\times\mathcal{D})_\Pi : \langle x,s\rangle \mapsto \langle (\mathcal{M}_D)_\Pi(x), \mathcal{D}_O(\mathcal{D}_I(s))\rangle$. It follows that $\mathcal{R}(\Pi, \mathcal{M}\times\mathcal{D}) = \mathcal{R}(\Pi, \mathcal{M})$. By (1.09), this is sufficient.

Corollary. If $\mathcal{M}$ is a device and $\mathcal{D}$ is a device with trivial input and output, then $\mathcal{M} <_m \mathcal{M}\times\mathcal{D}$, where $\mathcal{D}$ is not active for input or output in this product.

2.09. Theorem. If $\mathcal{M}$ is a machine and $\mathcal{D}$ is a trivial device, then $\mathcal{M} \sim_m \mathcal{M}\times\mathcal{D}$, where $\mathcal{D}$ is not active for input or output in this product.

Proof: (1.13, 2.08).

By (2.09), the machine conventions are not invalidated by use of variables as described at (1.14).

2.10. Theorem. If $\mathcal{M}$ is a machine, then $\mathcal{M} <_m \text{Aux}\times\mathcal{M}$.

Proof: Since $\text{Aux}_0(\text{Aux}_I(<>))=0$, (2.08) applies.

Corollary. If $\mathcal{M}$ is a machine and X is $\mathcal{M}$-acceptable, then X is $\text{Aux}\times\mathcal{M}$-acceptable.

2.11. Lemma. If $\mathcal{M}$ is a machine, then $\text{Aux}^2\times\mathcal{M} <_m \text{Aux}\times\mathcal{M}$.

Proof: Let Π be a program. Denote the commands for the two Aux components of $\text{Aux}^2\times\mathcal{M}$ by A_1, A_2. Obtain a program Π' from Π by replacing all occurrences of A_1, A_2 as commands by A. By induction on the length of computations, it is clear that

$$<\zeta,x_1,x_2,m>\ldots<\zeta',<>,<>,m'> \in \mathcal{C}(\Pi, \text{Aux}^2\times\mathcal{M}) \text{ for some } x_1,x_2$$

$$\Leftrightarrow <\zeta,x,m>\ldots<\zeta',<>,m'> \in \mathcal{C}(\Pi',\text{Aux}\times\mathcal{M}) \text{ for some } x.$$

It follows that $\mathcal{R}(\Pi',\text{Aux}\times\mathcal{M})= \mathcal{R}(\Pi,\text{Aux}^2\times\mathcal{M})$, as required.

2.12. Theorem. If $\mathcal{M}$ is a machine and $n\in\mathfrak{N}\setminus\{0\}$, then $\text{Aux}^n\times\mathcal{M} \sim_m \text{Aux}\times\mathcal{M}$.

Proof: By induction on n, using (2.10, 2.11).

We are now ready for our first applications of (1.17), leading to theorem (2.15), which generalizes the equivalence of Aux×In1-acceptable and Aux×Out-generable.

2.13. Lemma. If A is a finite set, $\mathcal{D}$ a device, and Π a program, then there is a program Π' such that $\text{Aux}\times\mathcal{D}\times\text{Out}^{(A)}_{\Pi'}(w,x,0)=<0,y,z>$ for some $w\in 2^*$ if and only if $\text{In1}^{(A)}\times\mathcal{D}_\Pi(z,x)=<0,y>$.

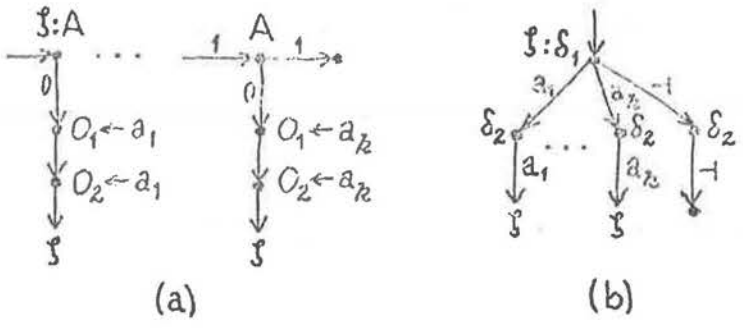


Figure 2. Programs in proofs of Lemmas 2.13 and 2.14.
(a) Φ in Lemma 2.13. (b) Ψ in Lemma 2.14.

Proof: (The double output trick) Let $A=\{a_1 \ldots, a_k\}$ and define Φ to be the program of figure 2(a). Clearly $\mathrm{ran}(\mathrm{Aux} \, (\mathrm{Out}^{(A)})^2_\Phi)=\{\langle 0,z,z\rangle \mid z\in A^*\}$. Application of (1.17) completes the proof.

2.14. Lemma. If A is a finite set, $\mathfrak{B}$ a device, and Π a program, then there is a program Π' such that $\mathrm{In1}^{(A)}\times\mathfrak{B}_{\Pi'}(z,x)=\langle 0,y\rangle$ if and only if $\mathfrak{B}\times\mathrm{Out}^{(A)}_\Pi(x,0)=\langle y,z\rangle$.

Proof: (The double input trick) Let $A=\{a_1 \ldots, a_k\}$ and define Ψ to be the program of figure 2(b). Clearly $\mathrm{dom}(\mathrm{In1}^{(A)})^2_\Psi=\{\langle z,z\rangle \mid z\in A^*\}$. Application of (1.17) completes the proof.

2.15. Theorem. If A is a finite set, $\mathfrak{M}$ a machine with $\mathfrak{M}_K=\mathfrak{M}_L=\Box$, and $X\subset A^*$, then X is $\mathrm{Aux}\times\mathrm{In1}^{(A)}\times\mathfrak{M}$-acceptable if and only if X is $\mathrm{Aux}\times\mathfrak{M}\times\mathrm{Out}^{(A)}$-generable.

Proof: (2.13, 2.11, 2.14).

The following notion, which is quite natural to consider, will also be useful.

2.16. If Σ is a countably infinite set, then S is a sequential relation

in Σ if and only if there are finite sets $A,B\subset\Sigma$ and a program Π such that $S=\mathcal{R}(\Pi,Aux\times Inl^{(A)}\times Out^{(B)})\subset A^*\times B^*$.

By way of (2.13, 2.14), (1.17) can again be applied to obtain the following.

2.17. Theorem. S is a sequential relation in a countably infinite set Σ if and only if the converse of S is also.

Proof: Let $S^{\smile}$ denote the converse of S and let $S=\mathcal{R}(\Pi,Aux\times Inl^{(A)}\times Out^{(B)})$.

In (2.13), take $\mathcal{D}=Aux\times Out^{(B)}$ and see there is a program Π' such that $Aux^2\times Out^{(B)}\times Out^{(A)}_{\Pi'}(w',w,0,0)=\langle 0,0,y,x\rangle$ for some $w'\in 2^*$ if and only if $Aux\times Inl^{(A)}\times Out^{(B)}_{\Pi}(w,x,0)=\langle 0,0,y\rangle$. By (2.11), it follows that there is a program Π'' such that ran $\mathcal{R}(\Pi'',Aux\times Out^{(B)}\times Out^{(A)})=S^{\smile}$.

In (2.14), take $\mathcal{D}=Aux\times Out^{(A)}$ and see there is a program Π''' such that $Aux\times Inl^{(B)}\times Out^{(A)}_{\Pi'''}(w,y,0)=\langle 0,0,x\rangle$ if and only if $Aux\times Out^{(B)}\times Out^{(A)}_{\Pi''}(w,0,0)=\langle 0,y,x\rangle$. Thus $\mathcal{R}(\Pi''',Aux\times Inl^{(B)}\times Out^{(A)})=S^{\smile}$.

Since $S=(S^{\smile})^{\smile}$, this completes the proof.

We begin now to obtain the characterization itself, starting with some basic definitions included here for completeness.

2.18. A family of languages is a pair $\langle\mathcal{L},\Sigma\rangle$ such that Σ is a countably infinite set and, for each $L\in\mathcal{L}$, there is some finite set $A\subset\Sigma$ with $L\subset A^*$. It is also required that $L\neq\emptyset$ for some $L\in\mathcal{L}$.

2.19. A family of languages $\langle\mathcal{L},\Sigma\rangle$ is closed under intersection with regular sets if and only if $L\cap R\in\mathcal{L}$ whenever $L\in\mathcal{L}$ and $R\in\mathfrak{R}_\Sigma$.

2.20. If A and B are sets, then a function $f:A^*\to B^*$ is a homomorphism

(of strings) if and only if domf=A* and f(xy)=f(x)f(y) for all x,yεA*.

Corollary 1. for f as above, f(<>)=<>.

Corollary 2. For A and B sets, f:A*→B* is a homomorphism if and only if there is a function $\hat{f}$:A→B* with dom$\hat{f}$=A such that $f(a_1 \ldots a_n)=\hat{f}(a_1)\ldots\hat{f}(a_n)$ for any $a_1 \ldots a_n \varepsilon A$.

For A,B,f as above, we employ the usual notations f(X), $f^{-1}(Y)$ thus:

If X⊂A*, then $f(X)=\{f(x) \mid x\varepsilon X\}$.

If Y⊂B*, then $f^{-1}(Y)=\{x \mid f(x)\varepsilon Y\}$.

2.21. A family of languages $<\mathcal{L},\Sigma>$ is closed under homomorphism [resp. inverse homomorphism] if and only if $f(L)\varepsilon\mathcal{L}$ [resp. $f^{-1}(L)\varepsilon\mathcal{L}$] whenever $L\varepsilon\mathcal{L}$ and f:A*→B* is a homomorphism with A,B finite subsets of Σ.

2.22. A family of languages $<\mathcal{L},\Sigma>$ is closed under sequential relations if and only if $\{y \mid xSy \text{ for some } x\varepsilon L\}\varepsilon\mathcal{L}$ whenever $L\varepsilon\mathcal{L}$ and S is a sequential relation in Σ.

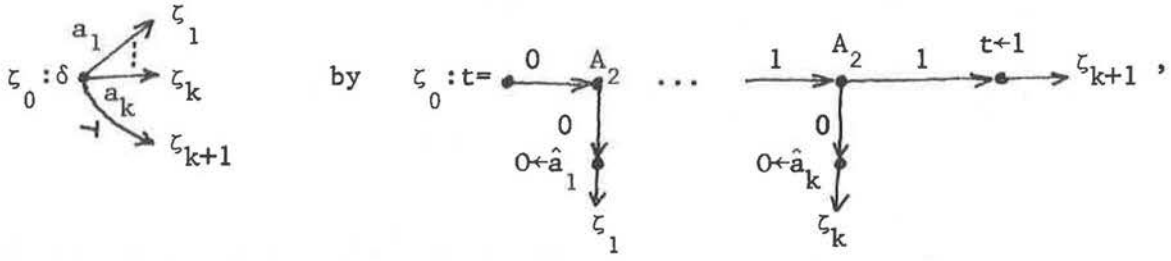
2.23. Theorem. A family of languages $<\mathcal{L},\Sigma>$ is closed under sequential relations if and only if it is closed under homomorphism, inverse homomorphism, and intersection with regular sets (if and only if it is a full trio, in the terminology of Ginsburg (1975)).

Proof: Suppose $<\mathcal{L},\Sigma>$ is closed under sequential relations. If f:A*→B* is a homomorphism with A and B finite subsets of Σ, then there is clearly a program Π such that $\mathcal{R}(\Pi, \mathrm{Inl}^{(A)}\times\mathrm{Out}^{(B)})=\{<x,f(x)> \mid x\varepsilon A^*\}$. By (2.10, 2.17), it follows that f(L), $f^{-1}(L)\varepsilon\mathcal{L}$.

If $R\varepsilon\mathfrak{R}_\Sigma$, then necessarily $R\varepsilon\mathfrak{R}_A$ for some finite set A⊂Σ. As remarked at (2.02), $R=\mathrm{dom}(\mathrm{Inl}^{(A)}_\Pi)$ for some program Π. Clearly a program Π' can be obtained

from Π with $\mathcal{R}(\Pi', \mathrm{In1}^{(A)} \times \mathrm{Out}^{(A)}) = \mathrm{Id}_R$. It follows that $L \cap R \in \mathcal{L}$ whenever $L \in \mathcal{L}$.

Conversely, suppose $\langle \mathcal{L}, \Sigma \rangle$ is closed under homomorphism, inverse homomorphism, and intersection with regular sets, and let $S = \mathcal{R}(\Pi, \mathrm{Aux} \times \mathrm{In1}^{(A)} \times \mathrm{Out}^{(B)})$ be a sequential relation in Σ. Let $a \mapsto \hat{a}$ be a one-to-one function with domain A and $\hat{A} = \{\hat{a} \mid a \in A\} \subset \Sigma$, $\hat{A} \cap B = \Box$. Obtain a program Π' from Π by replacing each



where t is a variable over 2 with initial value 0. Define homomorphisms

$g: (\hat{A} \cup B)^* \to A^*: \hat{a} \mapsto a, b \mapsto \langle\rangle$;

$f: (\hat{A} \cup B)^* \to B^*: \hat{a} \mapsto \langle\rangle, b \mapsto b$;

and see that $\{y \mid xSy \text{ for some } x \in L\} = f(g^{-1}(L) \cap \mathrm{ran}(\mathrm{Aux}^2 \times \mathrm{Out}^{(\hat{A} \cup B)}_{\Pi'})) \in \mathcal{L}$ whenever $L \in \mathcal{L}$.

2.24. If Σ is a countably infinite set and $\mathcal{M}$ is a machine with $\mathcal{M}_K = \mathcal{M}_L = \Box$, define the $\mathcal{M}$-family over Σ to be

$\hat{\mathcal{F}}(\mathcal{M}, \Sigma) = \{X \subset \Sigma^* \mid X \text{ is } \mathrm{Aux} \times \mathrm{In1}^{(A)} \times \mathcal{M}\text{-acceptable for some finite } A \subset \Sigma\}$.

Corollary 1. If $\mathcal{M}, \Sigma$ are as above, then $\hat{\mathcal{F}}(\mathcal{M}, \Sigma) = \{X \subset \Sigma^* \mid X \text{ is } \mathrm{Aux} \times \mathcal{M} \times \mathrm{Out}^{(A)}\text{-generable for some finite } A \subset \Sigma\}$. (By (2.15).)

Corollary 2. If $\mathcal{M}, \Sigma$ are as above, then $\langle \hat{\mathcal{F}}(\mathcal{M}, \Sigma), \Sigma \rangle$ is a family of languages. (Since $\{\langle\rangle\} = \mathrm{ran}\,\mathcal{R}(\mathrm{No}, \mathrm{Aux} \times \mathcal{M} \times \mathrm{Out}^{(A)})$.)

2.25. Examples. $\hat{\mathcal{F}}(\mathrm{Pds}^{(2)}, \Sigma)$ is the family of context-free languages over Σ, where Pds is as in example 1.18. $\mathcal{FA}_\Sigma = \hat{\mathcal{F}}(\mathrm{Id}, \Sigma)$, where Id is the device specified by $\mathrm{Id}_Q = \mathrm{Id}_S = \mathrm{Id}_T = 1$, $\mathrm{Id}_I = \mathrm{Id}_O : 0 \mapsto 0$, and $\mathrm{Id}_C = \Box$, not active for input or output.

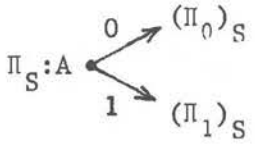
2.26. Theorem. If $\mathcal{M},\Sigma$ are as at (2.24), then $\langle\hat{\mathcal{F}}(\mathcal{M},\Sigma),\Sigma\rangle$ is closed under sequential relations.

Proof: (1.17, 2.11).

Corollary. If $\mathcal{M},\Sigma$ are as at (2.24), then $\langle\hat{\mathcal{F}}(\mathcal{M},\Sigma),\Sigma\rangle$ is a full trio.

2.27. Theorem. If $\mathcal{M},\Sigma$ are as at (2.24), then $\hat{\mathcal{F}}(\mathcal{M},\Sigma)$ is closed under union.

Proof: For i=0,1, let $X_i=\text{ran}\,\mathcal{R}(\Pi_i, \text{Aux}\times\mathcal{M}\times\text{Out}^{(A_i)})$. Obtain Π by adjoining

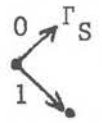


to the disjoint union (as programs) of Π_0 and Π_1. Clearly $\text{ran}\,\mathcal{R}(\Pi,\text{Aux}\times\mathcal{M}\times\text{Out}^{(A_0\cup A_1)})$ $=X_0\cup X_1$.

Corollary. In the terminology of Ginsburg (1975), $\langle\hat{\mathcal{F}}(\mathcal{M},\Sigma),\Sigma\rangle$ is a full semi-AFL.

2.28. Theorem. If $\mathcal{M},\Sigma$ are as at (2.24) with $m_0=(\mathcal{M}_D)_I(s)$ where $(\mathcal{M}_D)_S=\{s\}$, and for each program Π there is a program Γ (a reset program) such that $\langle\Gamma_S,m\rangle\ldots\langle 0,m_0\rangle\varepsilon\,\mathcal{C}_T(\Gamma,\mathcal{M}_D)$ whenever $\langle\Pi_S,m_0\rangle\ldots\langle\zeta,m\rangle\varepsilon\ \mathcal{C}(\Pi,\mathcal{M}_D)$, then $\hat{\mathcal{F}}(\mathcal{M},\Sigma)$ is closed under $^+$ (nontrivial concatenation closure).

Proof: Let $X=\text{ran}\,\mathcal{R}(\Pi,\text{Aux}\times\mathcal{M}\times\text{Out}^{(A)})$, and let Γ be as hypothesized, with respect to Π. Obtain Π' from the disjoint union (as programs) of Π and Γ by replacing each

ζ:• ($\varepsilon\Pi_T$) by 

and specifying $\Pi'_B(\zeta,i)=\Pi'_S=\Pi_S$ whenever $\Gamma_B(\zeta,i)=0\epsilon\Gamma_T$. Clearly ran $\mathcal{R}(\Pi',\text{Aux}\times\mathcal{M}\times\text{Out}^{(A)})=X^+$.

2.29. Theorem. A family of languages $<\mathcal{L},\Sigma>$ is closed under sequential relation, union, and $^+$ (is a full AFL, in the terminology of Ginsburg (1975)) if and only if there is a machine $\mathcal{M}$ satisfying the hypothesis of (2.28) with $\mathcal{L}=\hat{\mathcal{F}}(\mathcal{M},\Sigma)$.

Proof: By (2.26, 2.27, 2.28), $<\hat{\mathcal{F}}(\mathcal{M},\Sigma),\Sigma>$ is a full AFL.

Conversely, suppose $<\mathcal{L},\Sigma>$ is a full AFL. Define a device $\mathcal{D}$, not active for input or output, by

$\mathcal{D}_Q=\Sigma^*$, $\mathcal{D}_S=\mathcal{D}_T=1$.

$\mathcal{D}_I:0\mapsto<>$, $\mathcal{D}_O:<>\mapsto 0$ (undefined on nonempty strings).

$\mathcal{D}_C=\{W\leftarrow a \mid a\epsilon\Sigma\}+\mathcal{L}$.

$\mathcal{D}_{W\leftarrow a}:w\mapsto<wa,0>$.

$\mathcal{D}_L:w\mapsto<<>,0>$ if $w\epsilon L$, undefined otherwise.

A command of the last-mentioned type is an oracle for membership in $L\epsilon\mathcal{L}$.

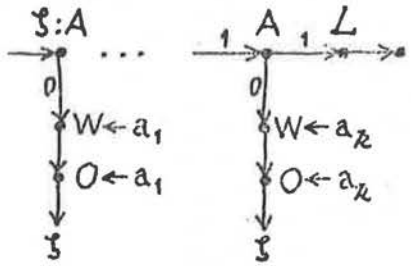


Figure 3. Program Π in proof of theorem 2.29.

If $L\epsilon\mathcal{L}$, then clearly $L=\text{ran}(\Pi,\text{Aux}\times\mathcal{D}\times\text{Out}^{(A)})$, where $A=\{a_1\ldots,a_k\}$, $L\subset A^*$, and Π is the program of figure 3. Thus $\mathcal{L}\subset\hat{\mathcal{F}}(\mathcal{D},\Sigma)$.

Finally, suppose Π is a program for $\text{Aux}\times\mathcal{D}\times\text{Out}^{(A)}$. Let

$$B=\{a\in\Sigma \mid W\leftarrow a \text{ occurs in } \Pi\},$$

$$C=\{L\in\mathcal{L} \mid L \text{ occurs (as a command) in } \Pi\}.$$

Let $L\mapsto\hat{L}$ be a one-to-one function with domain C and $\hat{C}=\{\hat{L}\mid L\in C\}\subset\Sigma$, $\hat{C}\cap B=\square$. Let $D=B\cup\hat{C}$. Obtain a program Π' for $\text{Aux}\times\text{Out}^{(D)}\times\text{Out}^{(A)}$ from Π by replacing each command $W\leftarrow a$ by $O_D\leftarrow a$ and each command L by $O_D\leftarrow\hat{L}$. (Commands $O\leftarrow a$ in Π are to be retained as $O_A\leftarrow a$.) Define $S=\text{ran}\,\mathcal{R}(\Pi',\text{Aux}\times\text{Out}^{(D)}\times\text{Out}^{(A)})$. By (2.14), $S=\mathcal{R}(\Pi'',\text{Aux}\times\text{In1}^{(D)}\times\text{Out}^{(A)})$ for some program Π', so S is a sequential relation in Σ. Clearly, $\text{ran}\,\mathcal{R}(\Pi,\text{Aux}\times\mathcal{D}\times\text{Out}^{(A)})=\{y\mid xSy \text{ for some } x\in(\cup\{L\{\hat{L}\}\mid L\in C\})^*\}\in\mathcal{L}$. Thus $\hat{\mathcal{F}}(\mathcal{D},\Sigma)\subset\mathcal{L}$.

$\mathcal{D}$ satisfies the hypotheses of (2.28) thus: If Π is a program for $\mathcal{D}$ and $B=\{a\in\Sigma\mid W\leftarrow a \text{ occurs in } \Pi\}$, then $\Gamma=$ is as required.

In the above proof, several facts about the full AFL $\langle\mathcal{L},\Sigma\rangle$ were used without proof: $\mathcal{L}$ is closed under concatenation and * (concatenation closure), and $\mathcal{R}_\Sigma\subset\mathcal{L}$. Proofs of these not using any development along the lines given here may be found in Ginsburg (1975).

REFERENCES

Baker, J. L. (1977). Simulation in a theory of programmable machines. University of British Columbia. Department of Computer Science. Technical Report 77-7.

Ginsburg, S. (1975). Algebraic and Automata-Theoretic Properties of Formal Languages. North-Holland. Amsterdam.

Knuth, D. E. (1975). Coroutines. The Art of Computer Programming, 1, 2nd edition. Section 1.4.2.