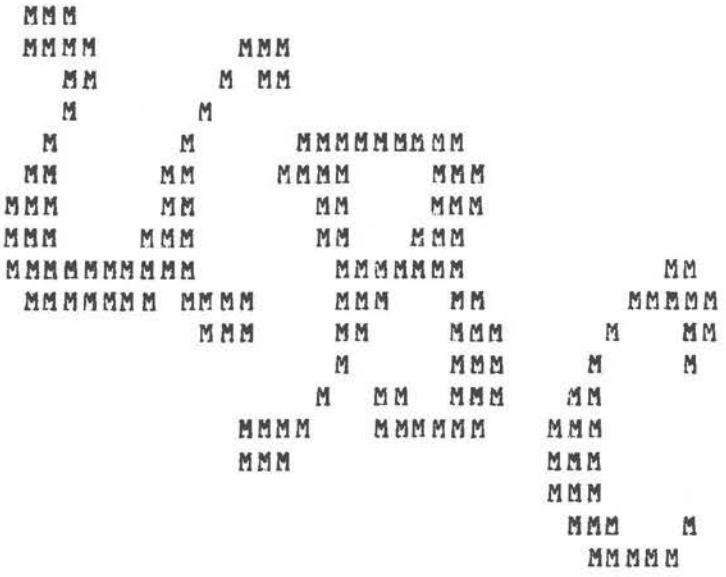


Simulation in a Theory of Programmable Machines

by

JOHN L. BAKER

Technical Report 77-7

July 1977

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

SIMULATION IN A THEORY OF PROGRAMMABLE MACHINES

John L. Baker

Department of Computer Science

University of British Columbia

ABSTRACT

In a theory of machines controlled by programs, automata-theoretic simulation can be presented simply and directly, and can be understood as an aspect of the algebraic structure of such machines.

Here the notions of product and homomorphism of devices in such a theory are presented, along with a notion of (computational) reducibility of one device to another. Simulation in the automata-theoretic sense is formally defined, and its validity as a technique for proving reducibility established uniformly.

The notions of device, product, homomorphism, and reducibility are then extended to model costs of computation evaluated a posteriori (in the manner of concrete complexity studies), and the validity of simulation as a proof technique established in this extended setting.

0. Introduction.

Many results in automata theory are justified by assertions of the form

(1) Any machine of species X can be simulated by some machine of species Y.

(For example, Hartmanis (1972) asserts that any (log n tape)-bounded turing machine can be simulated by some k-head automaton.) More precisely, an assertion of the form (1) means that for each formal specification A of species X there is a formal specification B of species Y and a function f from the set of states of A to the set of states of B such that, if one computational step according to A leads from a to a' , then there is a sequence of steps according to B leading from f(a) to f(a') , and (conversely) if no terminating computation by A begins with a , then no terminating computation by B begins with f(a) . For the purpose of such an assertion, a formal specification is given as a transition function or relation or a diagrammatic or tabular representation thereof, and includes definitions for "state", "computational step", and "terminating computation". (In practice, the simulation of (1) is usually uniform, so that the function f is independent of A , and effective, so that B can be constructed from A .)

Because of its directness and intuitive appeal, simulation would seem to be the first proof technique to which one would turn if he were seeking to establish the relative power of various computing resources as modeled by automata. Unfortunately, the appeal of simulation is very dependent on its conformity to a distinction between data and control structures--a distinction which is maintained in nearly all real-world programming, but is so badly represented by the formal specifications alluded to in the last paragraph that application of the simulation technique in a rigorous proof is impractical. The present state of affairs, then, is that many proofs in automata theory either avoid simulations at the expense of clarity or (more usually) use them at the expense of rigor.

The suggestion of Scott (1967), to present automata theory as a theory of programmable machines, overcomes this difficulty. In such a presentation, it is natural to specify a simulation of a machine X by a machine Y as comprising a state-representation function (as f above) together with a computational-step-representation function, whose values are state transformations of Y realizable

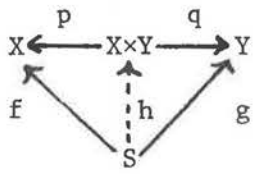
by microprograns simulating individual computational steps of X . If such a specification can be given, then the execution of a program on X can be simulated on Y by executing the same program, with individual commands interpreted as invocations of the appropriate simulating microprograms. The distinction between data and control structures which our intuition and experience call for is well enough respected by this way of specifying simulations that the contention between clarity and rigor for place in simulation-based proofs can be reduced to manageable proportions by its adoption.

The fiercest opponent of rigor is, of course, tediousness. No-one is glad to examine both the details of a simulation and a detailed proof that it is applicable, and this is cause enough that a sketch of the former is all that is usually offered. However, there is also a deeper reason for our reluctance to include proof of applicability with simulation arguments: Such proof seems redundant or out of place in the same way that proof that a particular group homomorphism preserves inverses would be. If simulation has any merit as a proof technique, it should be possible to establish its applicability uniformly--to show that, in some sense, any simulation which is well-defined on a set of generators extends necessarily to all the computations they generate. Unfortunately, the definitions of automata theory are not now given with enough uniformity to realize such a possibility. Scott's suggestions overcome this difficulty directly.

The difficulties described above as inherent in the usual presentation of automata theory can be expressed quite concisely: No algebraic structure for species of automaton has been developed, even though these are our primary models for data structures and their manipulation. In this paper, I present the elements of such an algebraic structure, that of programmable machines defined according to the suggestions of Scott (1967), and show how it can be used to define and apply simulations uniformly--how, in fact, simulation can be understood as an aspect of the algebraic structure of programmable machines.

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 $\{0\}$ (when convenient), and $\mathfrak{N}$ the set $\{0,1,2,3...\}$ of natural numbers.

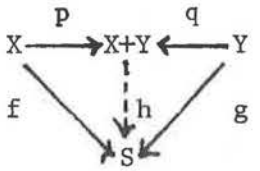
The cartesian product $X \times Y$ (together with projections p and q) is taken to be defined by the universal construction



.

That is, we use for $\langle X\times Y,p,q\rangle$ any triple with the property that, given a triple $\langle S,f,g\rangle$, where S is a set, f and g (total) functions, $f:S\to X, q:S\to Y$, there is a unique (total) function h such that $p\circ h=f$ and $q\circ h=g$. We say therefore that $X\times Y=Y\times X$ and $X\times 1=X$.

Likewise, the disjoint union X+Y is taken to be defined by the universal construction



,

so that we say $X+Y=Y+X$, $X+\square=X$, $X+Y=X\cup Y$ if $X\cap Y=\square$.

Infinite cartesian products and disjoint unions are treated similarly.

Apart from the above constructions, "function" here means "partial function". Specifically, $f:X\to Y$ means that f is a function (single-valued relation) defined for some elements of the set X and taking values in Y. $\mathrm{dom}f=\{x\,|\,y=f(x)$ for some $y\in Y\}$. $\mathrm{ran}f=\{y\,|\,y=f(x)$ for some $x\in X\}$. As usual, 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 $\mathrm{dom}(g\circ f)=\{x\,|\,f(x)\in\mathrm{dom}g\}$ and $\mathrm{ran}(g\circ f)=\{g(y)\,|\,y\in\mathrm{ran}f\}$.

In section 3, functions with numerical values are considered. With respect to these, the following conventions apply: If $f,g:X\to\mathbb{R}$, then $f+g:x\mapsto f(x)+g(x)$, with $\mathrm{dom}(f+g)=\mathrm{dom}f\cap\mathrm{dom}g$. $f\le g$ if and only if $\mathrm{dom}f=\mathrm{dom}g$ and $f(x)\le g(x)$ for all $x\in\mathrm{dom}f$. For $n\in\mathbb{R}$, n is also sometimes used to denote the function $x\mapsto n$.

If A is a set, then A^* denotes the set of strings (terminating sequences) in A. For $x \in A^*$, $|x|$ denotes the length of x (number of components). $(x]$ denotes the string obtained by deleting the first component, if any, of x. $[x)$ denotes the string obtained by deleting the last component, if any, of x. x^R denotes the reversal of x. $\langle\rangle$ denotes the empty string, ($|\langle\rangle|=0$, $(\langle\rangle]=[\langle\rangle)=\langle\rangle^R=\langle\rangle$.)

I am grateful for the financial assistance of the National Science Foundation of the United States* and the National Research Council of Canada† in developing the material presented here, as well as for the patience and interest of my students at the Universities of Calgary and British Columbia.

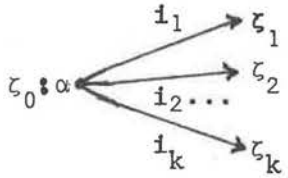
---

* Grant GJ-66, administered by Dr. Hellmut Golde.

† Grant A7882.

1. Programs, devices, products of devices.

Informally, this paper is about devices which execute programs. A program for a device $\mathcal{D}$ is a finite directed graph with labeled edges and nodes. (Loops and multiple edges are permitted.) A typical node ζ_0 has the appearance



Here, the label* α on ζ_0 is one of a set of commands valid for $\mathcal{D}$, and the labels $i_1, i_2, \ldots, i_k$ on edges directed from ζ_0 to $\zeta_1, \zeta_2, \ldots, \zeta_k$ are distinct elements of a set $\mathcal{D}_V(\alpha)$, the valence of α. A computation by Π on $\mathcal{D}$ is a sequence of pairs $\langle\zeta,m\rangle$, where ζ is a node of Π and m is an element of $\mathcal{D}_Q$, the memory set of $\mathcal{D}$. Associated with each command α is a partial function $\mathcal{D}_\alpha: \mathcal{D}_Q \to \mathcal{D}_Q \times \mathcal{D}_V(\alpha)$. Pairs $\langle\zeta,m\rangle$, $\langle\zeta',m'\rangle$ can occur consecutively in a computation by Π on $\mathcal{D}$ only if Π includes a node ζ_0 as above, $\zeta=\zeta_0$, and $\mathcal{D}_\alpha(m)=\langle m',i_j\rangle$ and $\zeta'=\zeta_j$ for some j. Execution of the program step specified at ζ_0, then, comprises modification of the memory configuration of $\mathcal{D}$ and (deterministic) selection of the next program step from those specified at $\zeta_1, \zeta_2, \ldots, \zeta_k$.

The specification of a device $\mathcal{D}$ also includes an input set $\mathcal{D}_S$, an input function $\mathcal{D}_I: \mathcal{D}_S \to \mathcal{D}_Q$, an output set $\mathcal{D}_T$, and an output function $\mathcal{D}_O: \mathcal{D}_Q \to \mathcal{D}_T$. A program node is terminal if it is not labelled with any command (and has no edges leading away from it). Each program Π has a specified start node Π_S. The

---

* Unfortunately, the sense in which the word "label" is used in the study of graphs conflicts with the sense in which it is used in the study of programming. To those who study graphs, the word denotes something that can occur more than once, like a particular opcode in a machine-language program. To those who study programming, the word denotes something which cannot meaningfully occur more than once, like (the name of) a particular node in a graph. I use "label" in the former sense in the first two paragraphs of this section, and avoid using it in the rest of the paper. It is good to think of the ζ_i as labels in the programming sense.

partial function computed by Π on $\mathfrak{D}$, $\mathfrak{D}_\Pi: \mathfrak{D}_S \to \mathfrak{D}_T$, is determined thus: For $x \in \mathfrak{D}_S$, if there is a computation by Π on $\mathfrak{D}$ starting with $\langle \Pi_S, \mathfrak{D}_I(x) \rangle$ and ending with some $\langle \zeta, m \rangle$ with ζ terminal, then $\mathfrak{D}_\Pi(x) = \mathfrak{D}_O(m)$. Otherwise, $\mathfrak{D}_\Pi$ is not defined at x.

The following is a more formal statement of our basic definitions.

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 $\zeta \in \Pi_Q$;

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

For example, supposing the node ζ_0 illustrated in the first paragraph of this section to occur in a program Π, we have $\Pi_A(\zeta_0) = \alpha$ and $\Pi_B(\zeta_0, i_1) = \zeta_1$, etc.

The usual dots-and-arrows notation for directed graphs is convenient for specifying programs. For a node ζ in a program Π: to specify that $\Pi_A(\zeta) = \alpha$, write "$\zeta:\alpha$" (or just "α" if no reference to ζ is needed) near the dot representing ζ; to specify that ζ is a terminal node, write "ζ:" or nothing near its dot; to specify that $\Pi_B(\zeta, i) = \zeta'$, write "i" near the arrow representing the appropriate edge $\langle \zeta, \zeta' \rangle$. Designate Π_S by putting its dot at the head of an arrow with nothing at its tail. To avoid graphic inconvenience, use an arrow with "ζ" but no

dot at its head to indicate that that arrow is to be taken as ending at the dot for node ζ.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

where $\langle \Pi_S, \mathcal{D}_I(x) \rangle \ldots \langle \zeta, m \rangle \varepsilon\ \mathcal{C}_T(\Pi, \mathcal{D})$ (uniquely, by (1.04)).

It is quite usual to specify automata as manipulating several data structures more or less independently--as having an input tape and one or more working tapes, counters, or pushdown stores, for example. It is easy to produce the same effect in a theory of programmable machines, and to do so in a way which shows the independence of the data structures very clearly: Define the <u>product</u> of a family $\mathcal{F}$ of devices to be a device whose memory, input, and output sets are the cartesian products of the corresponding sets in $\mathcal{F}$, and whose command set is the disjoint union of the command sets in $\mathcal{F}$. Define the input and output functions to operate componentwise, and each command interpretation to respond to and to affect only the component (of a memory set element) to which it applies.

The notion of product of devices is of great utility and fundamental importance. It permits the same input-output conventions to be applied to several data structures, or several input-output conventions to be applied to the same data structure. In particular, it permits non-determinism in string-based computability studies to be seen (as it ordinarily is in number-based computability studies) as a matter of projection--a relation computed non-deterministically as a projection of a function computed deterministically. Through this notion, the unifying power of Scott's suggestions is made more explicit.

Here is a slightly more formal definition, in which the cartesian product of a family $\{X_j \mid j \varepsilon J\}$ of sets (indexed by a set J) is taken to be the set of functions $f: J \to \bigcup_{j \varepsilon J} X_j$ with $f(j) \varepsilon X_j$ for all $j \varepsilon J$. (So in particular domf=J.)

<u>1.06</u>. If J is a set and for each $j \varepsilon J$ $\mathcal{D}_j$ is a device, then a device $\underset{j \varepsilon J}{\times} \mathcal{D}_j$ is defined thus (writing $\mathcal{P}$ for the <u>product</u> $\underset{j \varepsilon J}{\times} \mathcal{D}_j$):

$$\mathcal{P}_Q = \underset{j \varepsilon J}{\times} (\mathcal{D}_j)_Q, \quad \mathcal{P}_S = \underset{j \varepsilon J}{\times} (\mathcal{D}_j)_S, \quad \mathcal{P}_T = \underset{j \varepsilon J}{\times} (\mathcal{D}_j)_T.$$

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

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

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

For each $\alpha\varepsilon\mathcal{P}_C$, if $\alpha\varepsilon(\mathcal{D}_j)_C$, then $\mathcal{P}_\alpha$:m↦<m',i>, where m'=m except that m'(j) if such that $(\mathcal{D}_j)_\alpha(m(j))=<m'(j),i>$.

All products occurring in this paper have finite index set (J, in (1.06)). For these devices and for their input sets, etc., we use the usual notations $\mathcal{D}\times\mathcal{E}\times\ \ldots, \mathcal{D}^k$, leaving the specification of index set implicit. In case of repeated factors or where otherwise necessary, we distinguish commands by natural-number or other appropriate subscripts.

When a product is being denoted $\mathcal{D}\times\mathcal{E}\times\ldots$ or $\mathcal{D}^k$, elements of its input, output, and memory sets may be denoted by ordered tuples, the order corresponding to that of the notation for the product, but with singleton factors omitted.

It is worth remarking that a certain duality between data and control appears in (1.06). With respect to devices considered as data structures, the product construction is an inverse limit. With respect to programs, the associated control structures, the product construction is a direct limit.

The following examples illustrate the above definitions.

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

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

Inl_I:x↦x⊣, Inl_O:<>↦0 (undefined on non-empty strings).

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

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

This is, of course, a version of (one might say an interpreter for) the well-known finite automaton (Moore machine). The end marker $\dashv$ is necessary since the notion "final state" has been taken away from the device (and assigned to its programs). Notice that a terminating computation on In1 must include applications of δ at each position of its input and to the end marker, since $\mathrm{dom}\ \mathrm{In1}_0=\{\langle\rangle\}$.

1.08. Example. If A is a finite set and $k \in \mathfrak{N}\setminus\{0\}$, the k-head two-way input device $\mathrm{In2}^{(A,k)}$ is specified thus (writing K for $\mathrm{In2}^{(A,k)}$):

$K_Q=A^*\times\mathfrak{N}^k$, $K_S=A^*$, $K_T=1$.

$K_I: x \mapsto \langle x,0\ldots,0\rangle$, $K_0: \langle x,n_1\ldots,n_k\rangle \mapsto 0$.

$K_C= \{L_i \mid 1\le i\le k\}\cup\{R_i \mid 1\le i\le k\}\cup\{I_i= \mid 1\le i\le k\}$.

For each i, $1\le i\le k$,

$K_{L_i}: \langle x\ldots,n_i\ldots\rangle \mapsto \langle\langle x\ldots,n_i-1\ldots\rangle,0\rangle$ if $n_i>0$.

$K_{R_i}: \langle x\ldots,n_i\ldots\rangle \mapsto \langle\langle x\ldots,n_i+1\ldots\rangle,0$ if $n_i\le|x|$.

$K_{I_i=}: \langle x\ldots,n_i\ldots\rangle \mapsto \langle\langle x\ldots,n_i\ldots\rangle,a\rangle$, if, for some strings u,v over $A+\{\vdash,\dashv\}$, $x=uav$ and $|u|=n_i$. $(K_V(I_i=)=A+\{\vdash,\dashv\}.)$

The command interpretations K_α are undefined if the conditions stated are not met.

This device is also fairly familiar. It uses distinct heads to scan an input tape with end markers $\vdash$ and $\dashv$. The heads can be moved left or right independently, but not past the end markers. An $I_i=$ command controls branching in a program for $\mathrm{In2}^{(A,k)}$ according to the symbol scanned by the i^{th} head. Coincidence of heads is permitted but not generally detectable. Initially, all heads are scanning the left end marker $\vdash$.

The two-way input device $In2^{(A)}$ is defined to be $In2^{(A,1)}$. For this device, L, R, I= denote respectively L_1, R_1, $I_1=$.

1.09. Example. If B is a finite set, the turing device $Tur^{(B)}$ is specified thus (omitting the superscript):

$Tur_Q = B^* \times (B \cup \{<>\}) \times B^*$, $Tur_S = Tur_T = 1$.

$Tur_I : 0 \mapsto <<>,<>,<>>$, $Tur_O : <x,a,y> \mapsto 0$.

$Tur_C = \{L,R,T=\} \cup \{T \leftarrow a \mid a \in B\}$.

$Tur_L : <x,a,y> \mapsto <<[x),a',ay>,0>$ where $x = [x)a'$.

$Tur_R : <x,a,y> \mapsto <<xa,a',(y]>,0>$ where $y = a'(y]$.

$Tur_{T=} : <x,a,y> \mapsto <<x,a,y>,a>$. (So $Tur_V(T=) = B \cup \{<>\}$.)

$Tur_{T \leftarrow a'} : <x,a,y> \mapsto <<x,a',y>,0>$.

This is a version of the well-known turing machine. The memory element <x,a,y> represents the string xay over B scanned at the indicated occurrence of a. Possibly x=a=<>, so that the scan point may be beyond the left end of the string. Also possibly a=y=<>, so that the scan point may be beyond the right end. As for In2, the T= command reports the symbol (or <>) scanned. New information can be recorded by execution of a T←a command. In particular, execution of T←a on <<>,<>,y> or <x,<>,<>> lengthens the string. Initially, the string is empty. Execution of L leaves <<>,<>,y> unchanged, and execution of R leaves <x,<>,<>> unchanged. (You might say the unrecorded part of the infinite tape is slippery.)

By itself, Tur can only compute two functions: $0 \mapsto 0$ and the empty function. Interest lies in its role in products such as $In1^{(A)} \times Tur^{(B)}$ and $In2^{(A)} \times Tur^{(B)}$. (The set of domains of functions computable on these products is of course the set of recursively enumerable (or grammatical, languages over A.)

1.10. Example. The counter device Ctr is specified thus:

$Ctr_Q=\mathfrak{N}$, $Ctr_S=Ctr_T=1$.

$Ctr_I:0\mapsto 0$. $Ctr_0:x\mapsto 0$.

$Ctr_C=\{X+,X-\}$.

$Ctr_{X+}:x\mapsto\langle x+1,0\rangle$.

$Ctr_{X-}:0\mapsto\langle 0,0\rangle, x+1\mapsto\langle x,1\rangle$.

This device represents the behavior of a single natural-number register, initialized to 0, which can be incremented or (in a single operation) tested against 0 and decremented. Like Tur, it is insignificant by itself.

1.11. Example. According to (1.06, 1.08, 1.09), $In2^{(A)}\times Tur^{(B)}$ is specified thus (writing X for this product);

$X_Q=(A^*\times\mathfrak{N})\times(B^*\times(B\cup\{\langle\rangle\})\times B^*)$, $X_S=A^*$, $X_T=1$.

$X_I:x\mapsto\langle\langle x,0\rangle,\langle\langle\rangle,\langle\rangle,\langle\rangle\rangle\rangle$, $X_0:\langle\langle x,n\rangle,\langle u,a,v\rangle\rangle\mapsto 0$.

$X_C=\{L_I,R_I,I=,L_T,R_T,T=\}\cup\{T\leftarrow a\mid a\in B\}$.

$X_{L_I}:\langle\langle x,n\rangle,\langle u,a,v\rangle\rangle\mapsto\langle\langle\langle x,n-1\rangle,\langle u,a,v\rangle\rangle,0\rangle$ if $n>0$.

...

$X_{L_T}:\langle\langle x,n\rangle,\langle u,a,v\rangle\rangle\mapsto\langle\langle\langle x,n\rangle,\langle[u),a',av\rangle\rangle,0\rangle$, where $u=[u)a'$.

...

1.12. Example. Figure 1 exhibits a program Π for $In2^{(\{a\})}\times Tur^{(\{0,1\})}$, specified by some named program fragments to be assembled by the reader (functioning as a macroinstruction processor) in the obvious way.

Since $x1^j\mapsto[x)10^j$ is the transformation $n\mapsto n+1$ in binary numerals, it is clear that $dom(In2^{(\{a\})}\times Tur^{(\{0,1\})}_{\Pi})=\{a^{(2^j)}\mid j\in\mathfrak{N}\}$. (In this program and in all that follow, completely redundant branch labels are omitted.)

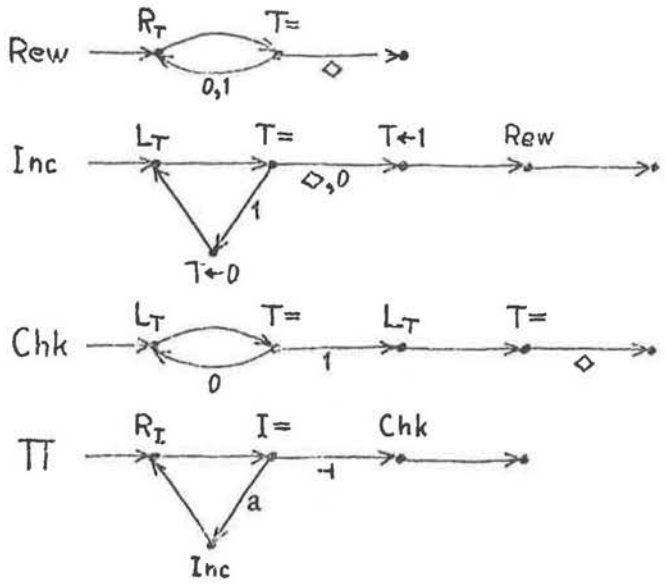


Figure 1. Program of example 1.12. On $\mathrm{Tur}^{(B)}$, Rew computes $\langle x,a,y\rangle \mapsto \langle xay,\langle\rangle,\langle\rangle\rangle$, Inc computes $\langle x1^j,\langle\rangle,\langle\rangle\rangle \mapsto \langle [x)10^j,\langle\rangle,\langle\rangle\rangle$, and Chk halts only on $\langle 10^j,\langle\rangle,\langle\rangle\rangle$.

1.13. Remark. The theory developed here follows the suggestion of Scott (1967) that computation should be deterministic by definition, non-determinism coming in only as a matter of interpretation. Specifically, a relation (non-deterministically) computed by a program Π on a device $\mathfrak{D}$ can be defined as the projection of the function $\mathfrak{D}_\Pi$ obtained by omitting a specified factor of $\mathfrak{D}_S$ (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 $\mathfrak{D}$ determined by the non-omitted components. It will be clear that the simulations studied here in terms of deterministic computation carry over usefully to the non-deterministic interpretation.

2. Reducibility, homomorphisms, and simulations.

One of the most important objectives of automata theory is the establishment of hierarchies of species of automaton according to their relative computational power. This is important because of the light it sheds on the relative intrinsic difficulty of computational problems.* In our theory of programmable machines, such hierarchies are represented as partial orderings of the class of devices. The following is perhaps the most obvious such ordering, and is definitive for comparison of computational power.

2.01. 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.

Corollary. $<$ as above is reflexive and transitive. $\sim$ as above is an equivalence relation.

To state some well-known facts in terms of this notion of reducibility, $\mathrm{Inl}^{(A)} \sim \mathrm{In2}^{(A)}$ for any finite set A, and $\mathrm{Inl}^{(A)} \times \mathrm{Tur}^{(B)} \not< \mathrm{Inl}^{(A)}$ if A and B

*According to informed opinion of the present day, the classification of problems according to species of automaton capable of their solution is crude or awkward enough that work on a classification according to the intrinsic difficulty of their solution on a particular, necessarily all-powerful, device (their complexity) is more likely to be fruitful. It remains true, however, that several fundamental distinctions (regularity of languages and computability itself, most importantly) appear very clearly in a species-of-automaton hierarchy, that species of automaton are more attractive than complexity classes per se as models for natural computing devices, and that it is known how to translate complexity (albeit crudely) into species of automaton, but not vice-versa. Furthermore, a well-formulated general notion of species of automaton may well be as useful to the study of a particular species as the general notion of field has been to the study of the real numbers. For all these reasons, it seems to me worthwhile to flaunt present opinion and devote some effort to alleviating the awkwardness of the species-of-automaton classification.

are non-empty. It is also true that $Tur^{(B)} \sim Ctr$, but only because only the trivial functions Id_1 and $\square$ can be computed on either of these devices.

The last remark suggests that, although reducibility is fundamental notion, it is not sharp enough to probe the structure of products. The following more restrictive ordering is useful in this connection.

2.02. If $\mathcal{D}$ and $\mathcal{D}'$ are devices, then $\mathcal{D} << \mathcal{D}'$ ($\mathcal{D}$ is stably reducible to $\mathcal{D}'$) if and only if $\mathcal{D} \times \mathcal{E} < \mathcal{D}' \times \mathcal{E}$ for all devices $\mathcal{E}$.

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

Corollary 1. << as above is reflexive and transitive. << as above is an equivalence relation.

Corollary 2. If $\mathcal{D}$ and $\mathcal{D}'$ are devices, then

(i) $\mathcal{D} \sim\sim \mathcal{D}'$ if and only if $\mathcal{D} \times \mathcal{E} \sim \mathcal{D}' \times \mathcal{E}$ for all devices $\mathcal{E}$.

(ii) If $\mathcal{D} << \mathcal{D}'$, then $\mathcal{D} < \mathcal{D}'$.

(iii) If $\mathcal{D} \sim\sim \mathcal{D}'$, then $\mathcal{D} \sim \mathcal{D}'$.

(To prove (ii), notice that $\mathcal{D} \sim \mathcal{D} \times Id$, where $Id_Q = Id_S = Id_T = 1$, $Id_I = Id_O = Id_1$, and $Id_C = \square$.

By way of example, it should be clear that $In1^{(A)} << In2^{(A)}$ for any finite set A (this will be proved in example 2.22), but $In2^{(A)}$ is not stably reducible to $In1^{(A)}$ if A is non-empty. Likewise, it should be clear (by an informal appeal to Church's thesis) that $Ctr << Tur$ and $Tur^n << Tur$ for any $n \in \mathfrak{N}$, where Tur denotes $Tur^{(B)}$ for some non-empty finite set B. We will show below that $Tur << Ctr^4$, whence it follows that $Tur \sim\sim Ctr^4$.

Any partial ordering can be regarded as a representation of the algebraic

structure of the underlying set*. However, such external representations are generally shallow. Deeper understanding comes from representations of internal structure, such as the following.

2.03. If $\mathcal{D}$ and $\mathcal{E}$ are devices with $\mathcal{D}_C \subset \mathcal{E}_C$, then a homomorphism f from $\mathcal{D}$ to $\mathcal{E}$ comprises functions $f_Q: \mathcal{D}_Q \to \mathcal{E}_Q$, $f_S: \mathcal{D}_S \to \mathcal{E}_S$, and $f_T: \mathcal{D}_T \to \mathcal{E}_T$ satisfying:

$\mathrm{dom} f_Q = \mathcal{D}_Q$.

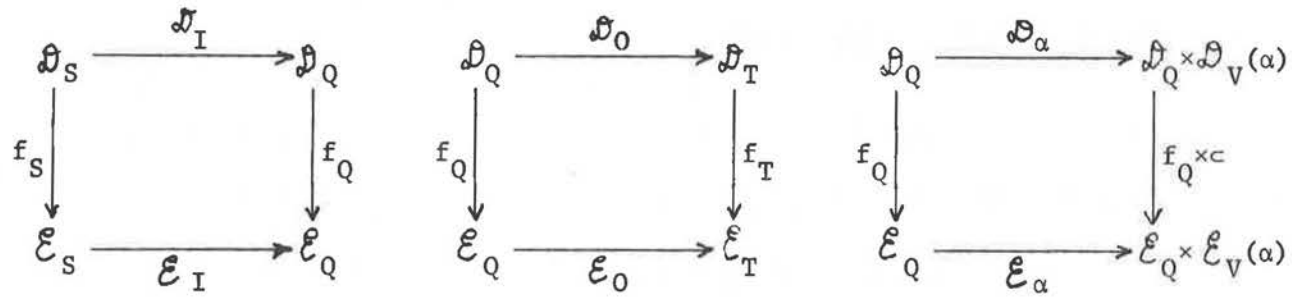
$f_Q \circ \mathcal{D}_I = \mathcal{E}_I \circ f_S$, $f_T \circ \mathcal{D}_O = \mathcal{E}_O \circ f_Q$.

$$\mathcal{E}_G(\alpha, f_Q(m)) = \begin{cases} \langle f_Q(m'), i\rangle \text{ if } \mathcal{D}_G(\alpha,m) = \langle m', i\rangle \\ \text{undefined if } \langle \alpha, m\rangle \notin \mathrm{dom}\, \mathcal{D}_G. \end{cases}$$

The last condition entails $\langle \alpha, f_Q(m)\rangle \in \mathrm{dom}\, \mathcal{E}_G \Leftrightarrow \langle \alpha, m\rangle \in \mathrm{dom}\, \mathcal{D}_G$ and $\mathcal{D}_V(\alpha) \subset \mathcal{E}_V(\alpha)$ for all $\alpha \in \mathcal{D}_C$. This condition can thus be stated $(f_Q \times c) \circ \mathcal{D}_G = \mathcal{E}_G \circ (c \times f_Q)$, or equivalently

$$(f_Q \times c) \circ \mathcal{D}_\alpha = \mathcal{E}_\alpha \circ f_Q \text{ for all } \alpha \in \mathcal{D}_C.$$

It is required, that is, that the following diagrams commute:



This notion of homomorphism is restrictive enough that it extends naturally to computations and thence to functions computed by a program, as we now show.

---

*In the language of category theory, just regard $\mathcal{D} < \mathcal{E}$ or $\mathcal{D} = \mathcal{E}$ as a morphism $\mathcal{D} \to \mathcal{E}$.

2.04. Lemma. If $\mathcal{D}$ and $\mathcal{E}$ are devices, f a homomorphism from $\mathcal{D}$ to $\mathcal{E}$, and Π a program, then

(i) If $\langle\zeta_0,m_0\rangle\ldots\langle\zeta_k,m_k\rangle\in\mathcal{C}(\Pi,\mathcal{D})$, then $\langle\zeta_0,f_Q(m_0)\rangle\ldots\langle\zeta_k,f_Q(m_k)\rangle\in\mathcal{C}(\Pi,\mathcal{E})$.

(ii) If $\langle\zeta_0,n_0\rangle\ldots\langle\zeta_k,n_k\rangle\in\mathcal{C}(\Pi,\mathcal{E})$, $\Pi_A(\zeta_j)\in\mathcal{D}_C$ for $0\le j<k$, and $n_0=f_Q(m_0)$ for some $m_0\in\mathcal{D}_Q$, then there are $m_1,\ldots,m_k\in\mathcal{D}_Q$ with $\langle\zeta_0,m_0\rangle\ldots\langle\zeta_k,m_k\rangle\in\mathcal{C}(\Pi,\mathcal{D})$ and $n_j=f_Q(m_j)$ for $1\le j<k$.

Proof: (i) is easily proved by induction on k. Also proceed by induction on k to prove (ii), which is trivially true for k=0. For k>0, let $\alpha=\Pi_A(\zeta_0)$. We have $\alpha\in\mathcal{D}_C\subset\mathcal{E}_C$, $\mathcal{D}_V(\alpha)\subset\mathcal{E}_V(\alpha)$. Thus $\mathcal{E}_\alpha(n_0)=\langle f_Q(m_1),i\rangle$ and $\mathcal{D}_\alpha(m_0)=\langle m_1,i\rangle$ for some $m_1\in\mathcal{D}_Q$, $i\in\mathcal{D}_V(\alpha)$. Application of the inductive hypothesis completes the proof.

2.05. Theorem. If $\mathcal{D}$ and $\mathcal{E}$ are devices, f a homomorphism from $\mathcal{D}$ to $\mathcal{E}$, and Π a program with $\Pi_C\subset\mathcal{D}_C$, then $\mathcal{E}_\Pi\circ f_S=f_T\circ\mathcal{D}_\Pi$.

Proof: Let $x\in\mathrm{dom}\,\mathcal{D}_\Pi$, $\langle\Pi_S,\mathcal{D}_I(x)\rangle\ldots\langle\zeta,m\rangle\in\mathcal{C}_T(\Pi,\mathcal{D})$. Then $\langle\Pi_S,f_Q(\mathcal{D}_I(x))\rangle\ldots\langle\zeta,f_Q(m)\rangle\in\mathcal{C}_T(\Pi,\mathcal{E})$ by (2.04(i)). Since $\mathcal{E}_I(f_S(x))=f_Q(\mathcal{D}_I(x))$, we have $\mathcal{E}_\Pi(f_S(x))=\mathcal{E}_0(f_Q(m))=f_T(\mathcal{D}_0(m))=f_T(\mathcal{D}_\Pi(x))$.

Conversely, if $f_S(x)\in\mathrm{dom}\,\mathcal{E}_\Pi$, $\langle\Pi_S,\mathcal{E}_I(f_S(x))\rangle\ldots\langle\zeta,n\rangle\in\mathcal{C}_T(\Pi,\mathcal{E})$, then, since $\mathcal{E}_I(f_S(x))=f_Q(\mathcal{D}_I(x))$ and $\Pi_C\subset\mathcal{D}_C$, there is some $m\in\mathcal{D}_Q$ such that $\langle\Pi_S,\mathcal{D}_I(x)\rangle\ldots\langle\zeta,m\rangle\in\mathcal{C}_T(\Pi,\mathcal{D})$ and $n=f_Q(m)$, by (2.04(ii)). Also, since $f_Q(m)\in\mathrm{dom}\,\mathcal{E}_0$, we have $m\in\mathrm{dom}\,\mathcal{D}_0$, so $x\in\mathrm{dom}\,\mathcal{D}_\Pi$.

The hypothesis $\Pi_C\subset\mathcal{D}_C$ in (2.05) is a technicality, and only necessary here because it was not made part of definitions 1.03 or 1.05. The weakness it induces is very slight, as we show in (2.07).

2.06. 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 $\zeta \in \Pi_Q$.

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

Proof: Let $\alpha \in \mathcal{D}_C$ be fixed, define $\Pi' = \Pi$ except that

(1) If $\Pi_A(\zeta) \notin \mathcal{D}_C$, set $\Pi'_A(\zeta) = \alpha$ and $\Pi'_V(\zeta) = \Box$.

(2) If $\Pi_A(\zeta) \in \mathcal{D}_C$, set $\Pi'_B(\zeta,i) = \Pi_B(\zeta,i)$ if $i \in \mathcal{D}_V(\Pi_A(\zeta))$, undefined otherwise.

2.08. Theorem. If $\mathcal{D}$ and $\mathcal{E}$ are devices, f a homomorphism from $\mathcal{D}$ to $\mathcal{E}$, and Π a program, then there is a program Π' such that $\mathcal{E}_{\Pi'} \circ f_S = f_T \circ \mathcal{D}_\Pi$.

Proof: If $\mathcal{D}_C = \Box$, then either $\mathcal{D}_\Pi = \Box$, in which case $f_T \circ \mathcal{D}_\Pi = \Box \circ f_S$, or $\mathcal{D}_\Pi = \mathcal{D}_O \circ \mathcal{D}_I$, in which case $f_T \circ \mathcal{D}_\Pi = \mathcal{E}_O \circ \mathcal{E}_I \circ f_S$. Both $\Box$ and $\mathcal{E}_O \circ \mathcal{E}_I$ are computable on $\mathcal{E}$.

If $\mathcal{D}_C \neq \Box$, then (2.05,2.07) apply to complete the proof.

We can now see that the existence of a homomorphism f from $\mathcal{D}$ to $\mathcal{E}$ with f_S and f_T suitably trivial implies that $\mathcal{D}$ is reducible to $\mathcal{E}$. In fact, this result extends to stable reducibility, as we now show.

2.09. Lemma. If $\mathcal{D}$ is a device and $e_\iota = Id_{\mathcal{D}_\iota}$ for $\iota \in \{Q,S,T\}$, then e is a homomorphism from $\mathcal{D}$ to $\mathcal{D}$.

2.10. Lemma. If J is a set, for each $j \in J$ $\mathcal{D}_j$ and $\mathcal{E}_j$ are devices with $(\mathcal{D}_j)_C \subset (\mathcal{E}_j)_C$ and f_j a homomorphism from $\mathcal{D}_j$ to $\mathcal{E}_j$, and $h_\iota = \underset{j\in J}{\times} (f_j)_\iota$ (that is, $h_\iota : t \mapsto t'$, where $t'(j) = (f_j)_\iota(t(j))$.) for $\iota \in \{Q,S,T\}$, then h is a homomorphism from $\underset{j\in J}{\times} \mathcal{D}_j$ to $\underset{j\in J}{\times} \mathcal{E}_j$.

Proof: We assume that foundations are so arranged that $\underset{j\in J}{+} (\mathcal{D}_j)_C \subset \underset{j\in J}{+} (\mathcal{E}_j)_C$. With this assumption, the proof is routine.

2.11. Theorem. If $\mathcal{D}$ and $\mathcal{D}'$ are devices and f a homomorphism from $\mathcal{D}$ to $\mathcal{D}'$, then, whenever $\mathcal{E}$ is a device and Π a program, there is a program Π' such that $(\mathcal{D}'\times\mathcal{E})_{\Pi'}\circ(f_S\times Id_{\mathcal{E}_S})=(f_T\times Id_{\mathcal{E}_T})\circ(\mathcal{D}\times\mathcal{E})_{\Pi}$.

Proof: For $\iota\in\{Q,S,T\}$, define $g=f_\iota\times Id_{\mathcal{E}_\iota}$. By (2.09, 2.10), g is a homomorphism from $\mathcal{D}\times\mathcal{E}$ to $\mathcal{D}'\times\mathcal{E}$. (2.08) completes the proof.

Corollary. If $\mathcal{D}$ and $\mathcal{D}'$ are devices and there is a homomorphism f from $\mathcal{D}$ to $\mathcal{D}'$ with $f_S=Id_{dom\,\mathcal{D}'_I}$ and $f_T=Id_{ran\,\mathcal{D}_O}$, then $\mathcal{D}\ll\mathcal{D}'$.

We come now to the definition of simulation, the principal subject of this paper. As will be seen, the notion of homomorphism is technically quite useful in our development. However, it is so restrictive that existence of a homomorphism from $\mathcal{D}$ to $\mathcal{E}$ is unusual even when $\mathcal{D}\ll\mathcal{E}$. This is because existence of a homomorphism from $\mathcal{D}$ to $\mathcal{E}$ implies that every step of a computation on $\mathcal{D}$ can be imitated on $\mathcal{E}$ as a single step. In a simulation, this condition is relaxed to permit $\mathcal{E}$ to imitate a single step of $\mathcal{D}$ by executing a sequence of steps--a program, in fact. This relaxation makes simulation much more useful for establishing reducibility between devices.

From an algebraic point of view, we may say that homomorphism reveals the internal structure of devices with respect to execution of single commands, so that it is quite insensitive to the interaction of programs and devices, whereas simulation reveals the internal structure of devices with respect to execution of programs. As we will see, existence of homomorphism implies existence of simulation, which (under suitable circumstances) implies stable reducibility. This heirarchy may be seen as proceeding from internal to external with respect to algebraic structure. Thus, we may regard simulation as revealing a peculiarly computational intermediate level of structure in devices.

2.12. If $\mathcal{D}$ and $\mathcal{E}$ are devices, then a simulation of $\mathcal{D}$ by $\mathcal{E}$ comprises functions $f_Q: \mathcal{D}_Q \to \mathcal{E}_Q$, $f_S: \mathcal{D}_S \to \mathcal{E}_S$, $f_T: \mathcal{D}_T \to \mathcal{E}_T$, f_I and $f_O: \mathcal{E}_Q \to \mathcal{E}_Q$, and for each $\alpha \in \mathcal{D}_C$, $f_\alpha: \mathcal{E}_Q \to \mathcal{E}_Q \times \mathcal{D}_V(\alpha)$, all satisfying:

$\mathrm{dom} f_Q = \mathcal{D}_Q$.

$f_Q \circ \mathcal{D}_I = f_I \circ \mathcal{E}_I \circ f_S$, $f_T \circ \mathcal{D}_O = \mathcal{E}_O \circ f_O \circ f_Q$.

$(f_Q \times \mathrm{Id}_{\mathcal{D}_V(\alpha)}) \circ \mathcal{D}_\alpha = f_\alpha \circ f_Q$ for each $\alpha \in \mathcal{D}_C$.

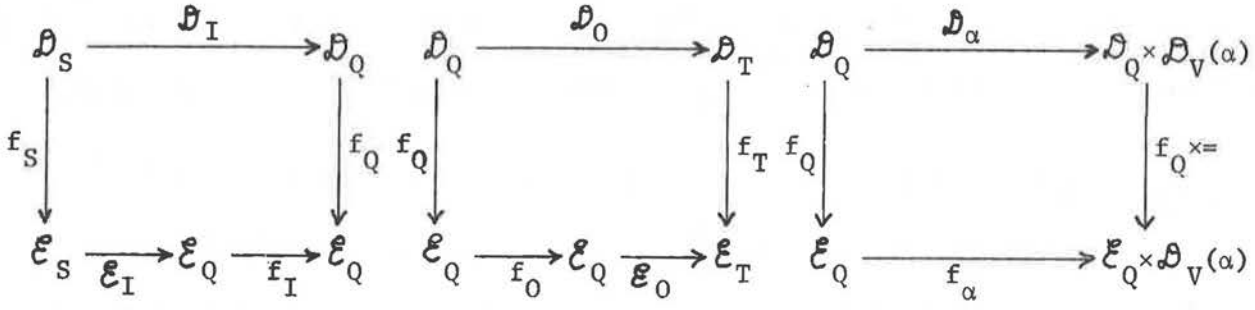
There are programs Γ_I and Γ_O such that, for $\iota \in \{I, O\}$,

$m' = f_\iota(m) \Leftrightarrow \langle (\Gamma_\iota)_S, m\rangle \ldots \langle 0, m'\rangle \in \mathcal{C}_T(\Gamma_\iota, \mathcal{E})$.

For each $\alpha \in \mathcal{D}_C$ and each finite subset $V \subseteq \mathcal{D}_V(\alpha)$, there is a program $\Gamma_\alpha^{(V)}$ such that

$(f_\alpha(m) = \langle m', i\rangle$ and $i \in V) \Leftrightarrow \langle (\Gamma_\alpha^{(V)})_S, m\rangle \ldots \langle i, m'\rangle \in \mathcal{C}_T(\Gamma_\alpha^{(V)}, \mathcal{E})$.

The strictly algebraic requirements of this definition are simply that the following diagrams commute:



The computational requirement is that $\mathcal{E}$ be microprogrammable to simulate $\mathcal{D}$ in the sense that f_I, f_O, and the f_α be computable on $\mathcal{E}$.

Since Π_T must be finite for any program Π, but $\mathcal{D}_V(\alpha)$ may be infinite, some stratagem like the inclusion of V as a parameter in the requirement that the $\Gamma_\alpha^{(V)}$ (supporting f_α) exist is necessary. If, as in the examples given below, $\mathcal{D}_V(\alpha)$ is finite, it is clearly sufficient to specify $\Gamma_\alpha^{(\mathcal{D}_V(\alpha))}$, which we denote simply

Γ_α.

Corollary. If $\mathcal{D}$ and $\mathcal{E}$ are devices and there is a homomorphism from $\mathcal{D}$ to $\mathcal{E}$, then there is a simulation of $\mathcal{D}$ by $\mathcal{E}$.

2.13. Example. If B is a finite set, then there is a simulation of $Tur^{(B)}$ by Ctr^4.

Proof: Everything is trivial if $B=\square$, so assume card $B=b>0$, $B=\{a_1,\ldots,a_b\}$. Define $w\mapsto\overline{w}:B^*\to\mathfrak{N}$ by $\overline{<>}=0$, $\overline{wa_i}=b\cdot\overline{w}+i$. (Thus $\overline{a_{i_1}\ldots a_{i_k}}=\sum_{j=1}^{k} i_j b^{j-1}$, so that w is the "b-adic" representation of $\overline{w}$.)

Define $f_S=f_T=Id_{\mathfrak{I}}$, $f_Q:<x,a,y>\mapsto<\overline{x},\overline{a},\overline{y^R},0>$, $f_I=f_0=Id_{\mathfrak{N}^4}$. For $\alpha\in Tur_C^{(B)}$, let f_α be defined to satisfy (2.12) with respect to the programs Γ_α given in figure 2. f is then the required simulation.

In the programs Γ_α in figure 2, the commands for the components of Ctr^4 are denoted L±, R±, C±, and U±, respectively. The circled characters show how to fill the ellipses. Γ_R is obtained from Γ_L by interchanging "L" and "R" throughout.

2.14. Example. For A a finite set, $t\in\mathfrak{N}$, define a device $IX4^{(A,t)}$ exactly as $In2^{(A)}\times Ctr^4$, but with $IX4^{(A,t)}_{X_i+}(w,n,x_1,x_2,x_3,x_4)$ undefined if $x_i\geq|\vdash w\dashv|^t-1$. (This is then $In2^{(A)}\times Ctr^4$, but with the Ctr components strictly bounded by $(|w|+2)^t$ on input w.)

It is easy to see that there is a simulation f of $IX4^{(A,t)}$ by $In2^{(A,4t+1)}$ with

$$f_Q:<w,n,x_1,x_2,x_3,x_4>\mapsto<w,n,d_{10},d_{11}..,d_{1,t-1},d_{20}\ldots,d_{30}\ldots,d_{40}\ldots d_{4,t-1}>,$$

where the d_{ij} are determined by

$$x_i=\sum_{j=0}^{t-1} d_{ij}\,|\vdash w\dashv|^j, \text{ each } d_{ij}$$

satisfying $0\leq d_{ij}<|\vdash w\dashv|$.

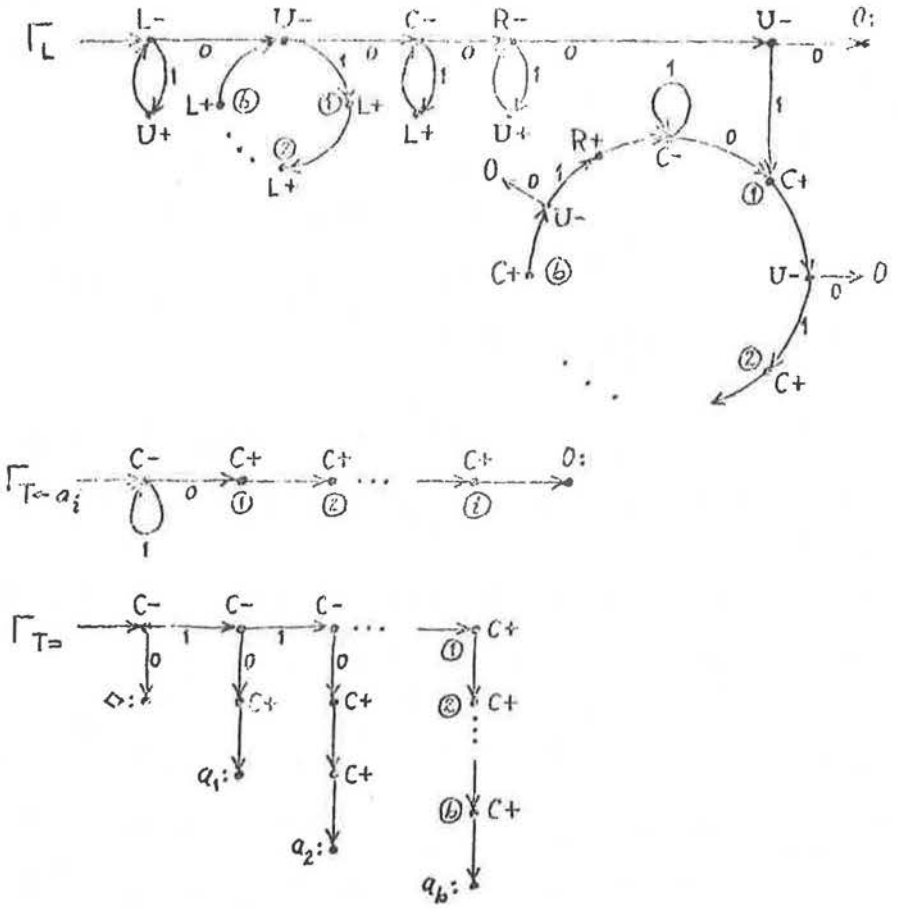


Figure 2. Programs supporting simulation f of $\mathrm{Tur}^{(B)}$ by Ctr^4 in example 2.13.

(Thus, in this simulation, $d_{i0}\ldots d_{i,t-1}$ is the $(|w|+2)$-ary representation of x_i, for x_i, for $i\varepsilon\{1,2,3,4\}$. This simple, if somewhat bizarre, example is useful in the proof of (3.18) below, which, like this example, is adapted from Hartmanis (1972).

It is intuitively obvious how, given a device $\mathcal{D}$, a program Π, and the microprograms Γ_α supporting a simulation of $\mathcal{D}$ by a device $\mathcal{E}$, to obtain a program Π' which makes $\mathcal{E}$ imitate $\mathcal{D}$ exactly: Simply regard each occurrence of a command α in Π as a macroinstruction whose expansion is Γ_α. The following theorem shows that such use of macroinstructions (supported by microprograms) in programs

for a device does not change its computing power. The proof given is an expression of the simple idea just mentioned, rather obscured by its devotion to avoiding technical pitfalls.

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

Proof: If $G = \Box$, then $\mathcal{D}' = \mathcal{D}$. If $\mathcal{D}_C = \Box$, then the set of functions computable on either $\mathcal{D}$ or $\mathcal{D}'$ is $\{\Box, \mathcal{D}_O \circ \mathcal{D}_I\}$. We therefore suppose $G \neq \Box$, $\mathcal{D}_C \neq \Box$.

If Π is a program, then there is a program Π' for $\mathcal{D}$ with $\mathcal{D}_{\Pi'} = \mathcal{D}_\Pi$, by (2.07). If Π' is for $\mathcal{D}$, we have $\mathcal{D}'_{\Pi'} = \mathcal{D}_{\Pi'}$. This shows $\mathcal{D} < \mathcal{D}'$.

Conversely, we proceed by induction on the number of nodes $\zeta \in \Pi_Q$ with $\Pi_A(\zeta) \in G$. If this is 0, then $\mathcal{D}_\Pi = \mathcal{D}'_\Pi$. Otherwise, let $\xi \in \Pi_Q$ with $\Pi_A(\xi) = \Gamma \in G$. If $\xi = \Pi_S$, $\Gamma_S \in \Gamma_T$, and $\Pi_B(\xi, \Gamma_S) = \xi$, then $\mathcal{D}_{\Pi'} = \Box$, and we may take $\Pi' = \rightarrow^\alpha$, where $\alpha \in \mathcal{D}_C$, to obtain $\mathcal{D}_{\Pi'} = \mathcal{D}'_\Pi$. Otherwise (the unexceptional case), by (2.07), assume Γ is for $\mathcal{D}$. Define a program Π' by:

$\Pi'_Q = (\Pi_Q \setminus \{\xi\}) + (\Gamma_Q \setminus \Gamma_T)$;

$\Pi'_A : \zeta \mapsto \begin{cases} \Pi_A(\zeta) & \text{if } \zeta \in \Pi_Q \\ \Gamma_A(\zeta) & \text{if } \zeta \in \Gamma_Q \end{cases}$;

$\Pi'_B : \langle \zeta, i \rangle \mapsto \begin{cases} \overline{\Pi_B(\zeta, i)} & \text{if } \zeta \in \Pi_Q \\ \overline{\Gamma_B(\zeta, i)} & \text{if } \zeta \in \Gamma_Q \end{cases}$;

$\Pi'_S = \overline{\Pi_S}$; all where

$\overline{\zeta} = \zeta$ if $\zeta \in \Pi'_Q$,

$\overline{\xi} = \begin{cases} \Gamma_S \text{ if } \Gamma_S \notin \Gamma_T \\ \Pi_B(\xi, \Gamma_S) \text{ if } \Gamma_S \in \Gamma_T \text{ and } \Pi_B(\xi, \Gamma_S) \neq \xi \\ \text{undefined if } \Gamma_S \in \Gamma_T \text{ and } \Pi_B(\xi, \Gamma_S) = \xi, \text{ and} \end{cases}$

$$\overline{\zeta}=\left\{\begin{array}{l}\Pi_B(\xi,\zeta) \text{ if } \Pi_B(\xi,\zeta)\neq\xi \\ \Gamma_S \text{ if } \Pi_B(\xi,\zeta)=\xi \text{ and } \Gamma_S\notin\Gamma_T \\ \text{undefined if } \Pi_B(\xi,\zeta)=\xi \text{ and } \Gamma_S\in\Gamma_T\end{array}\right\} \quad \text{if } \zeta\in\Gamma_T.$$

$\mathcal{D}'_{\Pi'}=\mathcal{D}'_{\Pi}$, and Π' has fewer G-nodes than Π, so the induction is established.

With the necessary tools in hand, we now establish analogues (2.16, 2.19) of (2.08, 2.11) with "simulation" replacing "homomorphism". By the corollary to (2.12), it would seem that the homomorphism results could be stated as corollaries. This is not the case for (2.08), however, since it is used in the proof of (2.16).

2.16. Theorem. If $\mathcal{D}$ and $\mathcal{E}$ are devices, f a simulation of $\mathcal{D}$ by $\mathcal{E}$, and Π a program, then there is a program Ψ such that $\mathcal{E}_\Psi\circ f_S=f_T\circ\mathcal{D}_\Pi$.

Proof: Define a device $\mathcal{D}'$ by $\mathcal{D}'_\iota=\mathcal{E}_\iota$ for $\iota\in\{Q,S,T\}$, $\mathcal{D}'_C=\mathcal{D}_C$, $\mathcal{D}'_I=f_I\circ\mathcal{E}_I$, $\mathcal{D}'_O=\mathcal{E}_O\circ f_O$, and $\mathcal{D}'_\alpha=f_\alpha$ for $\alpha\in\mathcal{D}_C$. Obviously f_Q,f_S,f_T make up a homomorphism from $\mathcal{D}$ to $\mathcal{D}'$. By (2.08), then, there is a program Π' such that $\mathcal{D}'_{\Pi'}\circ f_S=f_T\circ\mathcal{D}_\Pi$.

Define a device $\mathcal{D}''$ by $\mathcal{D}''_\iota=\mathcal{E}_\iota$ for $\iota\in\{Q,S,T,I,O\}$, $\mathcal{D}''_C=\{\Gamma_I,\Gamma_O\}\cup\{\Gamma_\alpha^{(V)}\mid\alpha\in\mathcal{D}_C$ and V is a finite subset of $\mathcal{D}_V(\alpha)\}$, $\mathcal{D}''_{\Gamma_\iota}:m\mapsto\langle f_\iota(m),0\rangle$ for $\iota\in\{I,O\}$, $\mathcal{D}''_{\Gamma_\alpha}(V):m\mapsto f_\alpha(m)$ if $f_\alpha(m)=\langle m',i\rangle$ and $i\in V$.

Define a program Π'' by $\Pi''_Q=\Pi'_Q+\{S,T\}$, $\Pi''_S=S$, $\Pi''_A(S)=\Gamma_I$, $\Pi''_B(S,0)=\Pi'_S$, $\Pi''_A(\zeta)=\Gamma_{\Pi'_A(\zeta)}^{(\Pi'_V(\zeta))}$ for $\zeta\in\Pi'_Q\backslash\Pi'_T$, $\Pi''_B(\zeta,i)=\Pi'_B(\zeta,i)$ for $\zeta\in\Pi'_Q\backslash\Pi'_T$ and $i\in\Pi'_V(\zeta)$, $\Pi''_A(\zeta)=\Gamma_O$ and $\Pi''_B(\zeta,0)=T$ for $\zeta\in\Pi'_T$. ($\Pi''_T=\{T\}$.) It is easy to see that, for all $x\in\mathrm{dom}\,\mathcal{E}_I$, $\langle S,\mathcal{E}_I(x)\rangle\langle\zeta_0,m_0\rangle\ldots\langle\zeta_k,m_k\rangle\langle T,\overline{m}\rangle\in\mathcal{C}_T(\Pi'',\mathcal{D}'')\Leftrightarrow(\langle\zeta_0,m_0\rangle\ldots\langle\zeta_k,m_k\rangle\in\mathcal{C}_T(\Pi',\mathcal{D}')$, $\zeta_0=\Pi'_S$, $m_0=\mathcal{D}'_I(x)$, and $\overline{m}=f_O(m_k)$.) It follows that $\mathcal{D}''_{\Pi''}=\mathcal{D}'_{\Pi'}$.

Finally, by (2.15), $\mathcal{D}''<\mathcal{E}$, so $\mathcal{E}_\Psi=\mathcal{D}''_{\Pi''}$ for some Ψ.

This proof displays the facility with which simulation can be approached using a program/device presentation. The data-structure aspect of the simulation, subsumed in our formal notion of simulation, is handled by the appeal to (2.15)

and the designation of ranf_Q as a distinguished set of simulating configurations in $\mathcal{E}_Q$ implicit in the definition of $\mathcal{D}'$. The construction of Π'' (renaming commands of Π' to indicate their reinterpretation as microprogram invocations and adding prelude Γ_I and postlude Γ_O) is all that is needed to handle the control-structure aspect of the simulation. Once these two aspects of simulation are clearly distinguished, each is easy to deal with, even in the most general setting.

2.17. Lemma. If $\mathcal{D}$ is a device, $e_\iota=\text{Id}_{\mathcal{D}_\iota}$ for $\iota\in\{Q,S,T\}$, $e_I=e_O=e_Q$, and $e_\alpha=\mathcal{D}_\alpha$ for all $\alpha\in\mathcal{D}_C$, then e is a simulation of $\mathcal{D}$ by $\mathcal{D}$.

2.18. Lemma. If J is a set, for each $j\in J$ $\mathcal{D}_j$ and $\mathcal{E}_j$ are devices and f_j a simulation of $\mathcal{D}_j$ by $\mathcal{E}_j$, and if $(f_j)_I\neq\text{Id}_{\text{ran}\,\mathcal{E}_I}$, $(f_j)_O\neq\text{Id}_{\text{dom}\,\mathcal{E}_O}$ for only finitely many $j\in J$, then h is a simulation of $\underset{j\in J}{\times}\mathcal{D}_j$ by $\underset{j\in J}{\times}\mathcal{E}_j$, where $h_\iota=\underset{j\in J}{\times}(f_j)_\iota$ for $\iota\in\{Q,S,T,I,O\}$ and, for $\alpha\in(\underset{j\in J}{\times}\mathcal{D}_j)_C$, if $m\in\underset{j\in J}{\times}(\mathcal{D}_j)_Q$ and $(f_j)_\alpha(m(j))=\langle\hat{m},i\rangle$, then $h_\alpha(m)=\langle m',i\rangle$ with $m'=m$ except that $m'(j)=\hat{m}$.

2.19. Theorem. If $\mathcal{D}$ and $\mathcal{D}'$ are devices and f a simulation of $\mathcal{D}$ by $\mathcal{D}'$, then, whenever $\mathcal{E}$ is a device and Π a program, there is a program Π' such that $(\mathcal{D}'\times\mathcal{E})_{\Pi'}\circ(f_S\times\text{Id}_{\mathcal{E}_S})=(f_T\times\text{Id}_{\mathcal{E}_T})\circ(\mathcal{D}\times\mathcal{E})_\Pi$.

Proof: By (2.17,2.18), there is a simulation g of $\mathcal{D}\times\mathcal{E}$ by $\mathcal{D}'\times\mathcal{E}$ with $g_S=f_S\times\text{Id}_{\mathcal{E}_S}$ and $g_T=f_T\times\text{Id}_{\mathcal{E}_T}$. (2.16) completes the proof.

Corollary. If $\mathcal{D}$ and $\mathcal{D}'$ are devices and there is a simulation f of $\mathcal{D}$ by $\mathcal{D}'$ with $f_S=\text{Id}_{\text{dom}\,\mathcal{D}'_I}$ and $f_T=\text{Id}_{\text{ran}\,\mathcal{D}_O}$, then $\mathcal{D}\ll\mathcal{D}'$.

This is the principal result of this paper. It shows that simulation can be used as intended in formal proofs establishing the stable reducibility of one device to another. By this result, for example, (2.13) is a rigorous proof that $\text{Tur}^{(B)}\ll\text{Ctr}^4$ for any finite set B.

The reader who is familiar with simulation arguments may be feeling uncomfortable about our requirement that a simulation be defined in terms of f_Q, which is a function and therefore single-valued. In many simulations, the representation of a simulated state depends not only on that state itself but also on the computation which led to it. A two-way tape simulation of a pushdown store, for example, may maintain its representation without erasing symbols removed from the pushdown store. This sort of consideration suggests that f_Q should be many-valued--that, in the definition of simulation, $\mathcal{E}_Q$ should be replaced by $\mathcal{E}_Q/R$, where R is a suitable equivalence relation.

Unfortunately, the specification of such a relation R corresponding to a particular simulation would likely be rather complicated, as would the proof that it is as required. This is because such an R must capture some important characteristics of computations, so cannot respect the separation of data and control structures as well as the present notion of simulation does.

Fortunately, the difficulty we are considering can be overcome without weakening the separation of data and control structures: If a simulation depends on computations as well as on configurations attained, simply define it as a simulation of a device which incorporates its computations in its configurations, but which cannot itself respond to the additional information its configurations contain. To complete this section, we give a uniform definition for such devices and show their simulation to be as useful for establishing reducibility as simulation of their precursors.

2.20. If $\mathcal{D}$ is a device, then a device $\mathrm{Rec}(\mathcal{D})$, the recording version of $\mathcal{D}$, is specified thus (writing R for $\mathrm{Rec}(\mathcal{D})$):

$R_Q=\{\langle\langle m_0,\alpha_0\rangle\ldots\langle m_{k-1},\alpha_{k-1}\rangle,m_k\rangle$ | For each j, $1\leq j\leq k$, there is some i such that

$\mathcal{D}_{\alpha_{j-1}}(m_{j-1})=<m_j,i>\}\subset(\mathcal{D}_Q\times\mathcal{D}_C)^*\times\mathcal{D}_Q$.

$R_S=\mathcal{D}_S$, $R_T=\mathcal{D}_T$.

$R_I:x\mapsto<<>,\mathcal{D}_I(x)>$, $R_O:<w,m>\mapsto\mathcal{D}_O(m)$.

$R_C=\mathcal{D}_C$.

For each $\alpha\in\mathcal{D}_C$, $R_\alpha:<w,m>\mapsto<<w<m,\alpha>,m'>,i>$, where $\mathcal{D}_\alpha(m)=<m',i>$.

Comparison of this definition with (1.03) shows that $<<m_0,\alpha_0>\ldots<m_{k-1},\alpha_{k-1}>,m_k>\in R_Q$ if and only if there is some program Π such that $<\zeta_0,m_0>\ldots<\zeta_k,m_k>\in\mathcal{C}(\Pi,\mathcal{D})$ and $\Pi_A(\zeta_j)=\alpha_j$ for all j, $0\leq j<k$. The name "recording version" for devices of this sort appeals to the fact that they (passively) retain a record of their previous states and state transitions, just as recording thermometers and the like do.

Corollary. If $\mathcal{D}$ and $\mathcal{E}$ are devices and Π a program, then $\mathcal{D}\times\mathcal{E}_\Pi=\mathrm{Rec}(\mathcal{D})\times\mathcal{E}_\Pi$. Thus $\mathcal{D}\sim\sim\mathrm{Rec}(\mathcal{D})$.

2.21. Theorem. If $\mathcal{D}$ and $\mathcal{D}'$ are devices and f a simulation of $\mathrm{Rec}(\mathcal{D})$ by $\mathcal{D}'$, then, whenever $\mathcal{E}$ is a device and Π a program, there is a program Π' such that $(\mathcal{D}'\times\mathcal{E}_{\Pi'})\circ(f_S\times\mathrm{Id}_{\mathcal{E}_S})=(f_T\times\mathrm{Id}_{\mathcal{E}_T})\circ(\mathcal{D}\times\mathcal{E}_\Pi)$.

Proof: By the corollary to (2.20), $\mathrm{Rec}(\mathcal{D})\times\mathcal{E}_\Pi=\mathcal{D}\times\mathcal{E}_\Pi$. (2.19) completes the proof.

Corollary. If $\mathcal{D}$ and $\mathcal{D}'$ are devices and there is a simulation f of $\mathrm{Rec}(\mathcal{D})$ by $\mathcal{D}'$ with $f_S=\mathrm{Id}_{\mathrm{dom}\,\mathcal{D}'_I}$ and $f_T=\mathrm{Id}_{\mathrm{ran}\,\mathcal{D}_O}$, then $\mathcal{D}<<\mathcal{D}'$.

2.22. Example. If A is a finite set, then there is a simulation of $\mathrm{Rec}(\mathrm{In1}^{(A)})$ by $\mathrm{In2}^{(A)}$.

Proof: Define $f_Q:((A+\{\dashv\})^*\times\{\delta\})^*\times(A+\{\dashv\})^*\to A^*\times\mathfrak{N}:<<>,x\dashv>\mapsto<x,0>$, $<<x\dashv,\delta>w,y>\mapsto<x,|x\dashv|-|y|>$. Define $f_S=\mathrm{Id}_{A^*}$, $f_T=\mathrm{Id}_1$, $f_I:A^*\times\mathfrak{N}\to A^*\times\mathfrak{N}:<x,0>\to<x,0>$, $f_O:A^*\times\mathfrak{N}\to A^*\times\mathfrak{N}:<x,|x|+1>\mapsto<x,|x|+1>$ (undefined otherwise). Define

$f_\delta: A^* \times \mathfrak{N} \to (A^* \times \mathfrak{N}) \times (A + \{\neg\}) : \langle x, n \rangle \mapsto \langle \langle x, n+1 \rangle, a \rangle$ if $n \leq |x|$, and, for some strings u, v, over $A+\{\neg\}$, $x = uav$ and $|u| = n$. Consideration of figure 3 shows that f is as required.

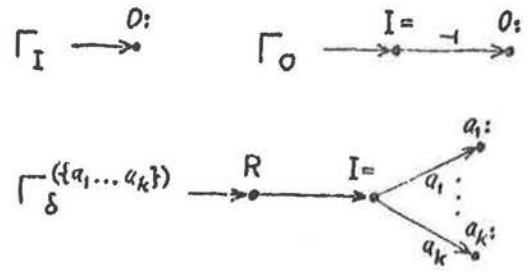


Figure 3. Programs supporting simulation f of $\mathrm{Rec}(\mathrm{In1}^{(A)})$ by $\mathrm{In2}^{(A)}$ in example 2.22.

By (2.21), this example proves that $\mathrm{In1}^{(A)} << \mathrm{In2}^{(A)}$ for any finite set A.

3. Cost schedules for devices.

Another important objective of automata theory is to provide models of computation which are suitable for presenting practical algorithms or important parts thereof. To be fully useful, such models must represent real-world computations in such a way that their costs--the demands they make on computing resources--can be estimated.

Both in theory and in practice, costs can be considered a priori: Devices can be defined to have limited resources, and costs modelled by hierarchies of devices. Likewise, real-world costs of computation can be limited by monitoring execution. This a priori approach in the theory of computing reduces cost estimation to computability, so that in our presentation it makes reducibility of one device to another the basis for cost comparisons, and simulation a proof technique of fundamental importance.

However, both practical and theoretical workers now usually consider cost a posteriori: A program is run and then the bill is paid; an algorithm is specified for execution in an environment of unlimited resources, and analysis shows the bounds on its demands. While not so fundamental as in the a priori approach, the role of simulation is important here too: It is often convenient to present algorithms in terms of distinct models for computation (one- and two-tape turing machines, for example) while retaining just one model as the basis for cost comparisons. To do this, it is sufficient to present simulations of the other models by the basic model, these simulations so constructed that costs are uniformly translated.

It is the work of this section to develop this a posteriori approach to cost in terms of our theory of programmable machines. We show how costs can be included in our model, how the notions of cost and reducibility interact, and how simulation can be used in this part of the theory.

3.01. If $\mathfrak{D}$ is a device, then a cost schedule c for $\mathfrak{D}$ comprises functions $c_I: \mathfrak{D}_S \to \mathfrak{N}$, $c_O: \mathfrak{D}_Q \to \mathfrak{N}$, and, for each $\alpha \in \mathfrak{D}_C$, $c_\alpha: \mathfrak{D}_Q \to \mathfrak{N}$, all such that dom c_ι = dom $\mathfrak{D}_\iota$ for $\iota \in \{I,O\} + \mathfrak{D}_C$.

3.02. If $\mathcal{D}$ is a device and c a cost schedule for $\mathcal{D}$, then $\langle \mathcal{D}, c \rangle$ is a device/cost pair and $\$(\mathcal{D},c)$ is a device, $\mathcal{D}$ operating at cost c, defined thus:

$\$(\mathcal{D},c)_Q = \mathcal{D}_Q \times \mathfrak{N}$, $\$(\mathcal{D},c)_S = \mathcal{D}_S$, $\$(\mathcal{D},c)_T = \mathcal{D}_T \times \mathfrak{N}$.

$\$(\mathcal{D},c)_I : x \mapsto \langle \mathcal{D}_I(x), c_I(x) \rangle$, $\$(\mathcal{D},c)_O : \langle m,j \rangle \mapsto \langle \mathcal{D}_O(m), j+c_O(m) \rangle$.

$\$(\mathcal{D},c)_C = \mathcal{D}_C$.

for each $\alpha \in \mathcal{D}_C$, $\$(\mathcal{D},c)_\alpha : \langle m,j \rangle \mapsto \langle \langle m', j+c_\alpha(m) \rangle, i \rangle$, where $\mathcal{D}_\alpha(m) = \langle m', i \rangle$.

With respect to a particular device/cost pair $\langle \mathcal{D}, c \rangle$, if Π is a program, then c_Π denotes the function $\mathcal{D}_S \to \mathfrak{N}$ determined by requiring that $\$(\mathcal{D},c)_\Pi = (\mathcal{D}_\Pi \times c_\Pi) \circ (x \mapsto \langle x,x \rangle)$.

3.03. Example. The cost schedule spc (space) for $\mathrm{Tur}^{(B)}$ is defined by $\mathrm{spc}_\iota = 0$ except that $\mathrm{spc}_{T \leftarrow a}(x,\langle\rangle,y) = 1$. Clearly, if $\langle \zeta, \langle x,a,y \rangle, k \rangle \ldots \langle \zeta', \langle x',a',y' \rangle, k' \rangle \in \mathcal{C}(\Pi, \$(\mathrm{Tur}, \mathrm{spc}))$, then $k'-k = |x'a'y'| - |xay|$.

It should be clear how to arrange a cost schedule to reflect costs which are, intuitively speaking, time-oriented: To charge for every step, for example, just set $c_\alpha = 1$ for all α. (3.03) shows how to arrange for costs which are not time-oriented: Arrange to detect the moment at which the cost is incurred. Even if the design of the device makes it difficult to detect that moment, it should be possible to achieve the desired effect with a cost schedule for the recording version, as in the following example.

3.04. Example. For $\langle w,m \rangle \in \mathrm{Rec}(\mathrm{Ctr}^4)_Q$, define $\|w,m\|$ to be the largest integer occurring in m ($\in \mathfrak{N}^4$) or in any component of w ($\in (\mathfrak{N}^4 \times \mathrm{Ctr}_C^4)^*$).

Define a cost schedule max for $\mathrm{Rec}(\mathrm{Ctr}^4)$ by $\max_\iota = 0$ except that $\max_{X_i \to} (w, \langle x_1, x_2, x_3, x_4 \rangle) = 1$ if $x_i = \|w, \langle x_1, x_2, x_3, x_4 \rangle\|$. Clearly, if $\langle \zeta, \langle w,m \rangle, k \rangle \ldots \langle \zeta', \langle w',m' \rangle, k' \rangle \in \mathcal{C}(\Pi, \$(\mathrm{Rec}(\mathrm{Ctr}^4), \max))$, then $k'-k = \|w',m'\| - \|w,m\|$.

We now extend the notions of reducibility, product, and stable reducibility to device/cost pairs, concluding with (3.09), which shows that applicability of simulation to these extended notions.

3.05. If $<\mathcal{D},c>$ and $<\mathcal{E},d>$ are device/cost pairs, then $<\mathcal{D},c><<\mathcal{E},d>$ ($<\mathcal{D},c>$ is reducible to $<\mathcal{E},d>$) if and only if $\$(\mathcal{D},c)<\$(\mathcal{E},d)$.

$<\mathcal{D},c>\sim<\mathcal{E},d>$ ($<\mathcal{D},c>$ is equivalent to $<\mathcal{E},d>$) if and only if $<\mathcal{D},c><<\mathcal{E},d>$ and $<\mathcal{E},d><<\mathcal{D},c>$, therefore if and only if $\$(\mathcal{D},c)\sim\$(\mathcal{E},d)$.

Corollary. $<$ as above is reflexive and transitive, $\sim$ as above is an equivalence relation.

3.06. If J is a set and for each $j\in J$ $<\mathcal{D}_j,c_j>$ is a device/cost pair, then a device/cost pair $\times_{j\in J}<\mathcal{D}_j,c_j>$ is defined as follows (writing $<\mathcal{P},c>$ for the product $\times_{j\in J}<\mathcal{D}_j,c_j>$), provided $\mathrm{domc}_I=\mathrm{dom}\,\mathcal{P}_I$ and $\mathrm{domc}_0=\mathrm{dom}\,\mathcal{P}_0$:

$$\mathcal{P}=\times_{j\in J}\mathcal{D}_j.$$

$$c_I:x\mapsto\sum_{j\in J}(c_j)_I(x(j)),\quad c_0:m\mapsto\sum_{j\in J}(c_j)_0(m(j)),$$

For each $\alpha\in\mathcal{P}_C$, if $\alpha\in(\mathcal{D}_j)_C$, then $c_\alpha:m\mapsto(c_j)_\alpha(m(j))$.

We may denote the cost schedule c in $<\mathcal{P},c>$ by $\sum_{j\in J}c_j$, which may be written $c_{j_1}+c_{j_2}+\ldots$.

With respect to a particular device $\mathcal{D}$, 0 denotes the cost schedule taking the value 0 for all valid arguments.

If $<\mathcal{D},c>$ is a device/cost pair and $\mathcal{E}$ a device, then $<\mathcal{D}\times\mathcal{E},c>$ denotes $<\mathcal{D},c>\times<\mathcal{E},0>$. Thus, with respect to such $\mathcal{D},c,\mathcal{E}$, c_Π denotes $(c+0)_\Pi$.

3.07. If $<\mathcal{D},c>$ and $<\mathcal{D}',c'>$ are device/cost pairs, then $<\mathcal{D},c><<<\mathcal{D}',c'>$ ($<\mathcal{D},c>$ is stably reducible to $<\mathcal{D}',c'>$) if and only if $<\mathcal{D},c>\times<\mathcal{E},d><<\mathcal{D}',c'>\times<\mathcal{E},d>$

for all device/cost pairs $\langle \mathcal{E}, d \rangle$.

$\langle \mathcal{D}, c \rangle \sim\sim \langle \mathcal{D}', c' \rangle$ ($\langle \mathcal{D}, c \rangle$ is stably equivalent to $\langle \mathcal{D}', c' \rangle$) if and only if $\langle \mathcal{D}, c \rangle <<< \langle \mathcal{D}', c' \rangle$ and $\langle \mathcal{D}', c' \rangle <<< \langle \mathcal{D}, c \rangle$.

3.08. Theorem. If $\langle \mathcal{D}, c \rangle$ and $\langle \mathcal{D}', c' \rangle$ are device/cost pairs, then $\langle \mathcal{D}, c \rangle <<< \langle \mathcal{D}', c' \rangle$ if and only if $\$(\mathcal{D}, c) << \$(\mathcal{D}', c')$. Therefore $\langle \mathcal{D}, c \rangle \sim\sim \langle \mathcal{D}', c' \rangle$ if and only if $\$(\mathcal{D}, c) \sim\sim \$(\mathcal{D}', c')$.

Proof: Suppose $\langle \mathcal{D}, c \rangle <<< \langle \mathcal{D}', c' \rangle$ and let $\mathcal{E}$ be any device. Clearly $\$(\mathcal{D} \times \mathcal{E}, c) \sim \$(\mathcal{D}, c) \times \mathcal{E}$ and $\$(\mathcal{D}' \times \mathcal{E}, c') \sim \$(\mathcal{D}', c') \times \mathcal{E}$. Since $\langle \mathcal{D} \times \mathcal{E}, c \rangle << \langle \mathcal{D}' \times \mathcal{E}, c \rangle$ by hypothesis, $\$(\mathcal{D}, c) \times \mathcal{E} << \$(\mathcal{D}', c') \times \mathcal{E}$ as required.

Conversely, suppose $\$(\mathcal{D}, c) << \$(\mathcal{D}', c')$ and let $\langle \mathcal{E}, d \rangle$ be any device/cost pair, Π any program. By hypothesis, there is a program Π' such that $\mathcal{D}' \times \mathcal{E}_{\Pi'} = \mathcal{D} \times \mathcal{E}_{\Pi}$, $\hat{c}'_{\Pi'} = \hat{c}_{\Pi}$, and $\hat{d}_{\Pi'} = \hat{d}_{\Pi}$, where $\hat{c}_{\Pi}, \hat{c}'_{\Pi'}$, $\hat{d}_{\Pi}$, and $\hat{d}_{\Pi'}$ are determined by requiring that $\$(\mathcal{D}, c) \times \$(\mathcal{E}, d)_{\Pi} = ((\mathcal{D} \times \mathcal{E})_{\Pi} \times \hat{c}_{\Pi} \times \hat{d}_{\Pi}) \circ h$ and $\$(\mathcal{D}', c') \times \$(\mathcal{E}, d)_{\Pi'} = ((\mathcal{D}' \times \mathcal{E})_{\Pi'} \times \hat{c}'_{\Pi'} \times \hat{d}_{\Pi'}) \circ h$, where $h: \langle x, y \rangle \mapsto \langle \langle x, y \rangle, x, y \rangle$. It follows that $(c+d)_{\Pi} = (c'+d)_{\Pi'}$, so $\langle \mathcal{D}, c \rangle \times \langle \mathcal{E}, d \rangle << \langle \mathcal{D}', c' \rangle \times \langle \mathcal{E}, d \rangle$, as required.

Corollary 1. $<<$ in the sense of (3.07) is reflexive and transitive. $\sim\sim$ in the sense of (3.07) is an equivalence relation.

Corollary 2. If $\langle \mathcal{D}, c \rangle$ and $\langle \mathcal{D}', c' \rangle$ are device/cost pairs, then

(i) $\langle \mathcal{D}, c \rangle \sim\sim \langle \mathcal{D}', c' \rangle$ if and only if $\langle \mathcal{D}, c \rangle \times \langle \mathcal{E}, d \rangle \sim\sim \langle \mathcal{D}', c' \rangle \times \langle \mathcal{E}, d \rangle$ for all device/cost pairs $\langle \mathcal{E}, d \rangle$.

(ii) If $\langle \mathcal{D}, c \rangle <<< \langle \mathcal{D}', c' \rangle$, then $\langle \mathcal{D}, c \rangle << \langle \mathcal{D}', c' \rangle$.

(iii) If $\langle \mathcal{D}, c \rangle \sim\sim \langle \mathcal{D}', c' \rangle$, then $\langle \mathcal{D}, c \rangle \sim\sim \langle \mathcal{D}', c' \rangle$.

3.09. Theorem. If $\langle \mathcal{D}, c \rangle$ and $\langle \mathcal{E}, d \rangle$ are device/cost pairs, then $\langle \mathcal{D}, c \rangle <<< \langle \mathcal{E}, d \rangle$ whenever there is a simulation f of $\mathcal{D}$ by $\mathcal{E}$ satisfying:

$f_S=Id_{dom\mathcal{E}_I}$, $f_T=Id_{ran\mathcal{D}_O}$,

$d_I+q_I\circ\mathcal{E}_I=c_I$, $(q_O+d_O\circ f_O)\circ f_Q=c_O$,

for each $\alpha\varepsilon\mathcal{D}_C$ and each finite subset $V\subset\mathcal{D}_V(\alpha)$, $q_\alpha^{(V)}\circ f_Q=c_\alpha$,

all where Γ_I, Γ_O, and $\Gamma_\alpha^{(V)}$ are programs as in (2.12) and each q is determined by

$$<\Gamma_S,\overline{m},0>\ldots<i,\overline{m}',q(\overline{m})>\varepsilon\,\mathcal{C}_T(\Gamma,\$(\mathcal{E},d))$$

for the corresponding Γ. (In particular, $domq_\iota=domf_\iota$ for $\iota\varepsilon\{I,O\}$, and $domq_\alpha^{(V)}=\{\overline{m}\mid f_\alpha(\overline{m})=<\overline{m}',i>$ for some $i\varepsilon V$ and some $\overline{m}'\}$.)

Proof: Define $\hat{f}_Q:<m,j>\mapsto<f_Q(m),j>$, $\hat{f}_S=f_S$, $\hat{f}_T:<y,j>\mapsto<f_T(y),j>$, $f_\iota:<\overline{m},j>\to<f_\iota(\overline{m}),\ j+q_\iota(\overline{m})>$ for $\iota\varepsilon\{I,O\}$, and, for each $\alpha\varepsilon\mathcal{D}_C$, $\hat{f}_\alpha:<\overline{m},j>\mapsto<\overline{m}',\ j+q_\alpha^{(\square)}(\overline{m}),i>$ if and only if $f_\alpha(\overline{m})=<\overline{m}',i>$. Then $\hat{f}$ is a simulation of $\$(\mathcal{D},c)$ by $\$(\mathcal{E},d)$ with $\hat{f}_S=Id_{dom\$(\mathcal{E},d)_I}$ and $\hat{f}_T=Id_{dom\$(\mathcal{D},c)_O}$. By the corollary to (2.19), $\$(\mathcal{D},c)<<\$(\mathcal{E},d)$. (3.08) completes the proof.

Like homomorphism, the above notion of reducibility is technically useful, but too restrictive to be the basis for a significant hierarchy. This is because costs must be carried over exactly in the relation $<\mathcal{D},c><<\mathcal{E},d>$. More significant results are obtained if d_Π, is merely subject to an upper bound determined by the corresponding c_Π. We proceed now to develop such a notion.

3.10. If X and Y are sets, then $\mathcal{F}(X,Y)$ denotes the set of (partial) functions $f:X\to Y$.

3.11. If $<\mathcal{D},c>$ and $<\mathcal{E},d>$ are device/cost pairs and $F:\mathcal{F}(\mathcal{D}_S,\mathfrak{N})\to\mathcal{F}(\mathcal{E}_S,\mathfrak{N})$, then $<\mathcal{D},c><<\mathcal{E},d>$ (F) ($<\mathcal{D},c>$ is reducible to $<\mathcal{E},d>$ at cost F) if and only if, whenever Π is a program, there is a program Π' such that $\mathcal{E}_{\Pi'}=\mathcal{D}_\Pi$ and $d_{\Pi'}\leq F(c_\Pi)$.

Corollary 1. If $\mathcal{D}$, $\mathcal{E}$, c, d, and F are as above and $<\mathcal{D},c><<\mathcal{E},d>(F)$, then $\mathcal{D}<\mathcal{E}$.

Corollary 2. (i) If $\langle \mathcal{D}_0, c_0 \rangle << \langle \mathcal{D}_1, c_1 \rangle$ (F) and $\langle \mathcal{D}_1, c_1 \rangle << \langle \mathcal{D}_2, c_2 \rangle$, then $\langle \mathcal{D}_0, c_0 \rangle << \langle \mathcal{D}_2, c_2 \rangle$(F).

(ii) If $\langle \mathcal{D}_0, c_0 \rangle << \langle \mathcal{D}_1, c_1 \rangle$ and $\langle \mathcal{D}_1, c_1 \rangle << \langle \mathcal{D}_2, c_2 \rangle$ (F), then $\langle \mathcal{D}_0, c_0 \rangle < \langle \mathcal{D}_2, c_2 \rangle$ (F).

3.12. If $\langle \mathcal{D}, c \rangle$ and $\langle \mathcal{D}', c' \rangle$ are device/cost pairs and $F: \mathcal{F}(\mathcal{D}_S; \mathcal{N}) \to \mathcal{F}(\mathcal{D}'_S; \mathcal{N})$, then $\langle \mathcal{D}, c \rangle <<< \langle \mathcal{D}', c' \rangle$(F) ($\langle \mathcal{D}, c \rangle$ is stably reducible to $\langle \mathcal{D}', c' \rangle$ at cost F) if and only if $\langle \mathcal{D} \times \mathcal{E}, c \rangle << \langle \mathcal{D}' \times \mathcal{E}, c' \rangle$ ($\hat{F}$) for all devices $\mathcal{E}$, where $\mathrm{dom}\hat{F} = \{f \circ (\langle x,y \rangle \mapsto x) \mid f \in \mathrm{dom}F\}$ and $\hat{F}: f \circ (\langle x,y \rangle \mapsto x) \mapsto F(f) \circ (\langle x,y \rangle \mapsto x)$.

(Equivalently, $\hat{F}$ is the restriction of $\mathcal{N}^{p'} \circ F \circ \mathcal{N}^{i}$ to $\{f \circ p \mid f \in \mathrm{dom}F\}$, where $p: \mathcal{D}_S \times \mathcal{E}_S \to \mathcal{D}_S: \langle x,y \rangle \mapsto x$, $p': \mathcal{D}'_S \times \mathcal{E}_S \to \mathcal{D}'_S: \langle x,y \rangle \mapsto x$, and $i: \mathcal{D}_S \to \mathcal{D}_S \times \mathcal{E}_S: x \mapsto \langle x, y_0 \rangle$, y_0 an arbitrary fixed element of $\mathcal{E}_S$.)

Corollary 1. If $\mathcal{D}, \mathcal{D}', c, c'$, and F are as above and $\langle \mathcal{D}, c \rangle <<< \langle \mathcal{D}', c' \rangle$(F), then $\mathcal{D} << \mathcal{D}'$ and $\langle \mathcal{D}, c \rangle << \langle \mathcal{D}', c' \rangle$(F).

Corollary 2. (i) If $\langle \mathcal{D}_0, c_0 \rangle <<< \langle \mathcal{D}_1, c_1 \rangle$(F) and $\langle \mathcal{D}_1, c_1 \rangle <<< \langle \mathcal{D}_2, c_2 \rangle$, then $\langle \mathcal{D}_0, c_0 \rangle <<< \langle \mathcal{D}_2, c_2 \rangle$(F).

(ii) If $\langle \mathcal{D}_0, c_0 \rangle <<< \langle \mathcal{D}_1, c_1 \rangle$ and $\langle \mathcal{D}_1, c_1 \rangle <<< \langle \mathcal{D}_2, c_2 \rangle$(F), then $\langle \mathcal{D}_0, c_0 \rangle <<< \langle \mathcal{D}_2, c_2 \rangle$(F).

Corollary 3. If $\langle \mathcal{D}, c \rangle$ is a device/cost pair and $F: \mathcal{F}(\mathcal{D}_S; \mathcal{N}) \to \mathcal{F}(\mathcal{D}_S; \mathcal{N})$ is such that $c_\Pi \leq F(c_\Pi)$ for all programs Π, then $\langle \mathcal{D}, c \rangle <<< \langle \mathcal{D}, c \rangle$(F). Therefore also $\langle \mathcal{D}, c \rangle << \langle \mathcal{D}, c \rangle$(F).

Corollary 4. If $\langle \mathcal{D}_0, c_0 \rangle << \langle \mathcal{D}_1, c_1 \rangle (F_1)$, $\langle \mathcal{D}_1, c_1 \rangle << \langle \mathcal{D}_2, c_2 \rangle (F_2)$, and $g \leq h \Rightarrow F_2(g) \leq F_2(h)$ whenever $g,h \in \mathrm{dom}F_2$, then $\langle \mathcal{D}_0, c_0 \rangle << \langle \mathcal{D}_2, c_2 \rangle (F_2 \circ F_1)$. Therefore also with "<<" in place of "<".

3.13. Theorem. If $\mathcal{D}, \mathcal{D}', c, c', F$, and $\hat{F}$ are as at (3.12), then

$\langle \mathcal{D},c\rangle <<< \langle \mathcal{D}',c'\rangle(F)$ if and only if, whenever $\langle \mathcal{E},d\rangle$ is a device/cost pair and Π a program, there is a program Π' such that $\mathcal{D}'\times\mathcal{E}_{\Pi'}=\mathcal{D}\times\mathcal{E}_{\Pi}$ and $(c'+d)_{\Pi'}\le\hat{F}(c_{\Pi})+d_{\Pi}$.

Proof: "If": Take d=0.

"Only if": We have $\langle \mathcal{D}\times\$(\mathcal{E},d),c\rangle << \langle \mathcal{D}'\times\$(\mathcal{E},d),c'\rangle(\hat{F})$. Thus, for some program Π', $\mathcal{D}'\times\mathcal{E}_{\Pi'}=\mathcal{D}\times\mathcal{E}_{\Pi}$, $c'_{\Pi'}\le\hat{F}(c_{\Pi})$, and $d_{\Pi'}=d_{\Pi}$. Since $(c'+d)_{\Pi'}=c'_{\Pi'}+d_{\Pi'}$, all is as required.

Corollary. If $\mathcal{D}$ is a device, $F:\mathcal{F}(\mathcal{D}_S,\mathcal{N})\to\mathcal{F}(\mathcal{D}_S,\mathcal{N})$, and c and c' are cost schedules for $\mathcal{D}$ such that $c'_{\Pi}\le F(c_{\Pi})$ for all programs Π, then $\langle \mathcal{D},c\rangle <<< \langle \mathcal{D},c'\rangle(F)$. Therefore also $\langle \mathcal{D},c\rangle << \langle \mathcal{D},c'\rangle(F)$.

3.14. Theorem. If $G,\mathcal{D}$, and $\mathcal{D}'$ are as at (2.15), c is a cost schedule for $\mathcal{D}$, and $c'_{\iota}=c_{\iota}$ for $\iota\in\{I,O\}+\mathcal{D}_c$ and $c'_{\Gamma}(m)=j \Leftrightarrow \langle\Gamma_S,m,0\rangle\ldots\langle\zeta,m',j\rangle\in\mathcal{C}_T(\Gamma,\$(\mathcal{D},c))$ for some m',ζ, then $\langle \mathcal{D},c\rangle\sim\sim\langle \mathcal{D}',c'\rangle$.

Proof: Let $\mathcal{E}$ be any device. Under the hypothesis, (2.15) shows that $\$(\mathcal{D},c)\times\mathcal{E}\sim\$(\mathcal{D}',c')\times\mathcal{E}$. That is, $\$(\mathcal{D},c)\sim\sim\$(\mathcal{D}',c')$. (3.08) completes the proof.

3.15. Theorem. If $\langle \mathcal{D},c\rangle$ and $\langle \mathcal{E},d\rangle$ are device/cost pairs and $F:\mathcal{F}(\mathcal{D}_S,\mathcal{N})\to\mathcal{F}(\mathcal{E}_S,\mathcal{N})$, then $\langle \mathcal{D},c\rangle <<< \langle \mathcal{E},d\rangle(F)$ whenever there is a simulation of $\mathcal{D}$ by $\mathcal{E}$ satisfying:

$f_S=\mathrm{Id}_{\mathrm{dom}f_I}$, $f_T=\mathrm{Id}_{\mathrm{ran}f_O}$, and

if Π is a program, $x\in\mathcal{D}_S$, $\langle\zeta_0,m_0\rangle=\langle\Pi_S,\mathcal{D}_I(x)\rangle$,

$\langle\zeta_0,m_0\rangle\ldots\langle\zeta_k,m_k\rangle\in\mathcal{C}_T(\Pi,\mathcal{D})$,

$V_j\subset\mathcal{D}_V(\Pi_A(\zeta_j))$ for all j, $0\le j<k$,

then $[d_I+q_I\circ\mathcal{E}_I](x)+[q_O+d_O\circ f_O](f_Q(m_k))+\sum_{j=0}^{k-1}q_j\le[F(c_{\Pi})](x)$,

where $q_j=q^{(V_j)}_{\Pi_A(\zeta_j)}(f_Q(m_j))$ for all j, $0\le j<k$,

all where q_I, q_O, and the $q_\alpha^{(V)}$ are defined as at (3.09) in terms of programs Γ_I, Γ_O, and $\Gamma_\alpha^{(V)}$ as in (2.12).

Proof: Notice that, if $m \in \mathcal{B}_Q$ and $m=m_j$ for some Π, x, and j as in the hypothesis above, then there is an upper bound on q_j considered as a function of V_j. In view of this, we can define a cost schedule $\overline{c}$ for $\mathcal{B}$ thus: $\overline{c}_I=d_I+q_I\circ\mathcal{E}_I$, $\overline{c}_O=(q_O+d_O\circ f_O)\circ f_Q$. For each $\alpha\in\mathcal{B}_C$,

$$\overline{c}_\alpha : m \mapsto \begin{cases} \max\{q_\alpha^{(V)}(f_Q(m)) \mid V \text{ finite, } V\subset \mathcal{B}_V(\alpha), \text{ and } f_Q(m)\in \mathrm{dom} q_\alpha^{(V)}\} \\ 0 \text{ if the above maximum does not exist, but } m\in \mathrm{dom}\,\mathcal{B}_\alpha. \end{cases}$$

By hypothesis, $\overline{c}_\Pi \le F(c_\Pi)$ for all programs Π, so by the corollary to (3.13), $\langle \mathcal{B}, c\rangle <<< \langle \mathcal{B}, \overline{c}\rangle (F)$.

Define a device $\mathcal{B}'$ by $\mathcal{B}_\iota' = \mathcal{E}_\iota$ for $\iota\in\{Q,S,T,I,O\}$, $\mathcal{B}'_C=\{\Gamma_I,\Gamma_O\}\cup\{\Gamma_\alpha^{(V)} \mid \alpha\in\mathcal{B}_C$ and V is a finite subset of $\mathcal{B}_V(\alpha)\}$, $\mathcal{B}'_{\Gamma_\iota} : \overline{m} \mapsto \langle f_\iota(\overline{m}), 0\rangle$ for $\iota\in\{I,O\}$, $\mathcal{B}'_{\Gamma_\iota(V)} : \overline{m}\mapsto f_\alpha(\overline{m})$ if $f_\alpha(\overline{m})=\langle \overline{m}', i\rangle$ and $i\in V$. Define a cost schedule c' for $\mathcal{B}'$ by $c'_\iota=d_\iota$, $c'_{\Gamma_\iota}=q_\iota$ for $\iota\in\{I,O\}$,

$$c'_{\Gamma_\alpha(V)} : \overline{m} \mapsto \begin{cases} \overline{c}_\alpha(m) \text{ if } \overline{m}=f_Q(m) \\ q_\alpha^{(V)}(\overline{m}) \text{ otherwise.} \end{cases}$$

$c'_{\Gamma_\alpha(V)}$ is well-defined since $\overline{c}_\alpha(m)=\overline{c}_\alpha(m')$ if $f_Q(m)=f_Q(m')$.) It is easy to see that f is a simulation of $\mathcal{B}$ by $\mathcal{B}'$ satisfying the hypotheses of (3.09) so that $\langle \mathcal{B}, \overline{c}\rangle <<< \langle \mathcal{B}', c'\rangle$.

Define a cost schedule c" for $\mathcal{B}'$ by c"=c' except that $c''_{\Gamma_\alpha(V)}=q_\alpha^{(V)}$ for all α,V. By the corollary to (3.13), $\langle \mathcal{B}', c'\rangle <<< \langle \mathcal{B}', c''\rangle$ $(\mathrm{Id}_{\mathcal{F}(\mathcal{B}_S,\mathcal{H})})$.

By (3.14), $\langle \mathcal{B}', c''\rangle \sim\sim \langle \mathcal{E}, c\rangle$.

Corollaries 2 and 4 of (3.12) complete the proof.

This is the principal result of this section. It shows how simulation can be used to establish stable reducibility at cost. It is worth noticing that the

proof of (3.15) is in close analogy to that of (2.16), while the result itself is analogous to the corollary to (2.19). Just as for the last-mentioned result, it is important to extend (3.15) to simulations of recording versions, as we now do.

3.16. Theorem. If $\mathcal{D}$,c, $\mathcal{E}$,d and F are as in (3.15), then $\langle\mathcal{D},c\rangle\langle\langle\langle\mathcal{E},d\rangle(F)$ whenever there is a simulation f of $\mathrm{Rec}(\mathcal{D})$ by $\mathcal{E}$ satisfying the conditions of (3.15), but with $\mathrm{Rec}(\mathcal{D})$ in the role of $\mathcal{D}$.

Proof: Define a cost function $\overline{c}$ for $\mathrm{Rec}(\mathcal{D})$ by $\overline{c}_I=c_I$, $\overline{c}_\iota:\langle w,m\rangle\mapsto c_\iota(m)$ for $\iota\in\{0\}+\mathcal{D}_C$. Define $w\mapsto\overline{w}:(\mathcal{D}_Q\times\mathcal{N}\times\mathcal{D}_C)^*\to(\mathcal{D}_Q\times\mathcal{D}_C)^*$ inductively by $\overline{\langle\rangle}=\langle\rangle$, $\overline{w\langle m,j,\alpha\rangle}=\overline{w}\langle m,\alpha\rangle$. Define $k_S=\mathrm{Id}_{\mathcal{D}_S}$, $k_T=\mathrm{Id}_{\mathcal{D}_T\times\mathcal{N}}$, $k_Q:(\mathcal{D}_Q\times\mathcal{N}\times\mathcal{D}_C)^*\times(\mathcal{D}_Q\times\mathcal{N})\to((\mathcal{D}_Q\times\mathcal{D}_C)^*\times\mathcal{D}_Q)\times\mathcal{N}:\langle w,\langle m,j\rangle\rangle\mapsto\langle\langle\overline{w},m\rangle,j\rangle$. k is a homomorphism from $\mathrm{Rec}(\$(\mathcal{D},c))$ to $\$(\mathrm{Rec}(\mathcal{D}),\overline{c})$, so, by the corollaries to (2.12) and (2.21), $\$(\mathcal{D},c)\langle\langle\$(\mathrm{Rec}(\mathcal{D}),\overline{c})$. By (3.08), then, $\langle\mathcal{D},c\rangle\langle\langle\langle\mathrm{Rec}(\mathcal{D}),\overline{c}\rangle$.

By (3.15), $\langle\mathrm{Rec}(\mathcal{D}),\overline{c}\rangle\langle\langle\langle\mathcal{E},d\rangle(F)$.

Corollary 2 to (3.12) completes the proof.

By way of illustration, the following examples apply several of the results presented above to obtain a very secure proof of the assertion of Hartmanis (1972) mentioned at the beginning of this paper.

3.17. Example. If B is a finite set, then $\langle\mathrm{Tur}^{(B)},\mathrm{spc}\rangle\langle\langle\langle\mathrm{Rec}(\mathrm{Ctr}^4),\max\rangle(g\mapsto(\mathrm{card}\ B)^{g+1})$.

Proof: Extend the definition of f in (2.13) in the obvious way to specify a simulation of $\mathrm{Rec}(\mathrm{Tur}^{(B)})$ by $\mathrm{Rec}(\mathrm{Ctr}^4)$ supported by the same programs Γ_α.

If $\langle\zeta_0,m_0\rangle=\langle\Pi_S,\mathrm{Rec}(\mathrm{Tur}^{(B)})_I(0)\rangle$ and $\langle\zeta_0,m_0\rangle\ldots\langle\zeta_k,m_k\rangle\in\mathcal{C}(\Pi,\mathrm{Rec}(\mathrm{Tur}^{(B)}))$, we have

$$[\max_I q_I \circ \mathrm{Rec}(\mathrm{Ctr}^4)_I](0)=0,$$

$$[q_0+\max_0 \circ f_0](f_Q(m_k))=0,$$

$$\sum_{j=0}^{k-1} q_j=||f_Q(m_k)||,$$

where q_I, q_0, q_j are as at (3.15), $||w,m||$ as at (3.04).

If $m_k=\langle w,\langle x,a,y\rangle\rangle$, we have

$$||f_Q(m_k)|| \leq \overline{xay}<b^{|xay|+1}=b^{\mathrm{spc}_\Pi(0)+1},$$

where b=card B, $\overline{w}$ are as in (2.13).

(3.16) completes the proof.

3.18. Example (Hartmanis, 1972). Let $L:\mathfrak{N}\to\mathfrak{N}:n\mapsto\lceil \lg(n+1)\rceil$. (lg = binary logarithm, $\lceil x\rceil$=least integer $\geq x$.)

For given finite sets A and B, there is an integer $k\in\mathfrak{N}$ such that, if Π is a program and

$$X=\mathrm{dom}(\mathrm{In2}^{(A)}\times\mathrm{Tur}^{(B)}_\Pi)\cap\{x\mid \mathrm{spc}_\Pi(x)\leq L(|x|)\},$$

then there is a program Ψ with $\mathrm{dom}(\mathrm{In2}^{(A,k)}_\Psi)=X$. (Notice that k is independent of Π.)

Proof: Modify Π so that, on input x, it first uses the counting technique of (1.12) to lay out a segment of tape with length $L(|x|)$ and then either carries out all its work within that segment or, if it cannot do so, fails to halt. Call the resulting program Π'. We have $X=\mathrm{dom}(\mathrm{In2}^{(A)}\times\mathrm{Tur}^{(B)}_{\Pi'})$ and $\mathrm{spc}_{\Pi'}(x)=L(|x|)$ for all $x\in X$.

By (3.17) and the corollary to (2.20), there is a program Π'' with $X=\mathrm{dom}(\mathrm{In2}^{(A)}\times\mathrm{Ctr}^4_{\Pi''})$ and $\max_{\Pi''}(x)\leq b^{L(|x|)+1}$ for all $x\in X$, where b=Card B. Without loss of generality, assume $b\geq 2$.

For $n>4$, $2\lg n>L(n)+1$, so $(n+2)^{2\lceil \lg b\rceil}>b^{L(n)+1}$. The latter relation also

holds for $n\varepsilon\{0,1,2,3,4\}$, so we have $\max_{\Pi''}(x)<(|x|+2)^{2\lceil \lg b\rceil}$ for all $x\varepsilon X$. Clearly then, $X=\mathrm{dom}(IX4^{(A,2\lceil \lg b\rceil)}_{\Pi''})$.

(2.14) and the corollary to (2.19) complete the proof, with $k=8\lceil \lg b\rceil+1$.

The a priori and a posteriori approaches to costs of computation are separable only as approaches--in practice and in theory both refer to the same matter, the allocation of computing resources. It is to be expected, therefore, that they will often appear together, as they do in the argument above.

REFERENCES

Hartmanis, J. (1972). On non-determinacy in simple computing devices. Acta Informatica. 1, 336:344.

Scott, D. (1967). Some definitional suggestions for automata theory. Journal of Computer and System Sciences. 1, 187:212.