10-1-1970

On Maximally Parallel Schemata

Robert M. Keller
Harvey Mudd College

Recommended Citation

Keller, R.M. "On maximally parallel schemata." Proceedings of the Eleventh Annual IEEE Symposium on Switching and Automata Theory (October 1970): 32-50. DOI: 10.1109/SWAT.1970.13

ON MAXIMALLY PARALLEL SCHEMATA

Robert M. Keller

Princeton University*
Princeton, New Jersey

Summary

A model for parallel computation called a schema is presented. This model is similar to that presented in the recent work of Karp and Miller[2]. Section 1 presents a description of the model, and some results on the characterization of computations within it. Section 2 summarizes some results on determinacy and equivalence. Section 3 presents a formalization of the property of maximal parallelism in schemata. Several alternate characterizations are shown to be equivalent for certain classes. Section 4 presents results on the complexity of a maximally parallel schema equivalent to a given schema.

Introduction

A mathematical model for parallel computation is the basis of this study. The definition of this model has been motivated by various proposed and existing methods for introducing parallelism into contemporary computing systems. By "parallelism" it is meant that several interacting processes may be simultaneously engaged in a computation.

The control of such parallel processes will be studied in the framework of parallel program schemata. The concept of a parallel program schema is derived from two historically distinct concepts: parallel program, and program schema.

The term "parallel program" was apparently first introduced by S. Gill[3], although the concept of parallelism had been used earlier. The use of special instructions for the control of parallel processes within programs appears first in the literature (to the author's knowledge) in Richards[4]. Similar instructions were called "fork" and "join" by Conway[5]. Instructions of a slightly different nature, such as "lock" and "unlock"[6] allow two strings to be executed in either order, but not simultaneously. Control of this type is called "non-persistent", in the terminology to be presented. In the present work, the concern will be with control of the persistent type. The model to be presented is of sufficient generality to make the use of specific instructions, such as fork and join, unnecessary. This allows a number of problems in specifying the control to be circumvented[7].

Given a way of expressing parallelism in programs, it is desirable to be able to convert a conventional program, i.e. one without explicit parallelism, to an equivalent parallel program. Such a conversion would permit the time required for a computation to be reduced, providing sufficient computational resources are available. This problem is of central interest here.

Bernstein[8] has observed that the problem of determining whether two consecutive blocks of a program can be executed in parallel is generally undecidable. This fact provides part of the motivation for introducing parallel program schemata, as will be seen.

The concept of a program schema was introduced by Yanov[9]. A program schema structurally resembles a program, but the specific functions associated with the elements of the program, e.g. the operations of addition, multiplication, etc., are replaced by abstract function-symbols. A program schema can therefore be thought of as a representation of a family of programs, each member of which is obtained by specifying functions in place of the abstract symbols.

The motivation for considering schemata is that they provide a way of simplifying analysis techniques, such as those required for removing inessential parts of programs. Moreover, they sometimes help avoid problems of undecidability. For example, it is well known that the problem of deciding whether two programs compute the same function is unsolvable. However it was shown by Yanov that the equivalence problem for his formulation of schemata is solvable. By equivalence of schemata it is meant that the programs resulting from assigning functions to the abstract operation symbols are equivalent, regardless of the particular assignment.

Recently various authors[10,11,12] have studied refinements of Yanov's original concept. These refinements differ essentially in the amount of information assumed about the memory. For example Yanov's formulation considers the entire memory as a single undifferentiated cell. The work of Luckham, Park and Paterson[10] allows the memory to be divided into a number of cells. It is notable that they showed that the equivalence problem is undecidable for schemata with two or more cells.

The schema concept can be combined with parallel programs in an effort to simplify the analysis of properties connected with parallelism. Work of this sort was first reported in Karp and Miller[2]. The authors show that parallel program schemata provide a fruitful approach to the problem of determinacy, a problem which does not exist when parallelism is absent. The schema approach will be shown to be useful in the present work in avoiding undeciable problems of the type observed by Bernstein.

* This research was completed while the author was an NSF Graduate Fellow at the University of California, Berkeley. The results presented here are taken from the author's doctoral dissertation[1].

Notation

Let $f: A \rightarrow B$ be a partial function. For any $a \varepsilon A$, $\ulcorner f(a) \urcorner$ means that f is defined for a. For $A' \subseteq A$ $f|_{A'}$ denotes the restriction of f to A'.

The set of all natural numbers $\{0,1,2,3,...\}$ is denoted by ω.

If Σ is any set, Σ^* denotes the set of all finite strings of elements in Σ. If $x,y \varepsilon \Sigma^*$, xy denotes the concatenation of x and y. The string of length 0 is denoted o.

Σ^ω denotes the set of all countably-infinite strings of elements in Σ. $\hat{\Sigma}$ denotes $\Sigma^* \cup \Sigma^\omega$. Concatenation is extended to map $\Sigma^* \times \hat{\Sigma} \rightarrow \hat{\Sigma}$ in the obvious fashion. If $x \varepsilon \Sigma^*$ and $y \varepsilon \hat{\Sigma}$, $x \leq y$ means $\exists z \varepsilon \hat{\Sigma}$ $x=yz$. $x \hat{<} y$ means $x \leq y$ and $x \neq y$.

For any $x \varepsilon \hat{\Sigma}$ and $n \varepsilon \omega$, if $|x| \geq n$, let ${}_n x$ denote the first n components of x, i.e. the $y \varepsilon \Sigma^n$ such that $y \leq x$. x_n denotes the n-th component of x.

For any $x \varepsilon \hat{\Sigma}$, $\sigma \varepsilon \Sigma$, define $\sigma \varepsilon x$ (σ "occurs in" x) if $\exists y \varepsilon \Sigma^*$ $\exists z \varepsilon \hat{\Sigma}$ $x=y\sigma z$. The same symbol will be used for set membership. The specific usage should be clear from context.

Let $\Sigma' \subseteq \Sigma$. For any $x \varepsilon \hat{\Sigma}$, let $E(\Sigma',x)$ denote the string x' obtained by deleting all occurrences of elements of $\Sigma - \Sigma'$ from x.

1. Parallel Program Schemata

This section presents the basic definitions and results for the model, to be used throughout the remainder of this work. Of particular concern will be the method of controlling parallel computations and the characterization of these computations in terms of strings of abstract symbols.

It should be noted that the definitions to be presented are not necessarily the most general. Indeed they represent a compromise of the desire to give the model sufficient generality to yield non-trivial results, yet not so general as to make the exposition cumbersome. An attempt is made to model those aspects which are felt significant in studying parallelism, as opposed to some other aspect of an algorithmic process.

1.1 Schemata and Interpretations

Definition An operation set is a finite set $A=\{a,b,c,....\}$ of elements called operations, together with the following for each $a \varepsilon A$:

(1) a unique symbol $\bar{a}$ called the initiator of a
(2) a non-empty finite set of unique symbols $\underline{a}=\{a_1,a_2,....,a_{K(a)}\}$ called terminators of a
(3) a finite set $D(a) \subset \omega$ called the domain of a
(4) a non-empty finite set $R(a) \subset \omega$ called the range of a

In addition, for any $B \subseteq A$ define $\bar{B}=\{\bar{a} \mid a \varepsilon B\}$, $\underline{B}= \bigcup_{a \varepsilon B} \underline{a}$, and $\Sigma = \bar{A} \cup \underline{A}$. For any $\sigma \varepsilon \Sigma$ let $\langle\sigma\rangle$ be the operation $a \varepsilon A$ such that $\sigma=\bar{a}$ or $\sigma \varepsilon \underline{a}$.

An interpretation for an operation set A is a quadruple $I=(\nabla,d_o,F,G)$ where

(1) ∇ is a set called the universe
(2) $d_o \varepsilon \nabla^\omega$ is called the initial assignment
(3) for each $a \varepsilon A$

$$F_a:\nabla^{D(a)} \rightarrow \nabla^{R(a)} \text{ is a total function}$$

$$G_a:\nabla^{D(a)} \rightarrow \underline{a} \text{ is a total function}$$

Int(A) will denote the class of all interpretations for A.

A schema is a pair $S=(A,T)$ where A is an operation set and $T=(Q,q_o,f,\phi)$ is the transducer, where

(1) Q is a countable set of states
(2) $q_o \varepsilon Q$ is the initial state
(3) $f: Q \times \underline{A} \rightarrow Q$ is a partial function, the state transition function
(4) $\phi: Q \rightarrow 2^A$ is a total function, the output function

In addition, every transducer must satisfy the following:

Axiom 1 $\forall q \varepsilon Q$ $\forall \sigma \varepsilon \underline{A}$ $\ulcorner f(q,\sigma) \urcorner$ iff $\langle\sigma\rangle \varepsilon \phi(q)$.

Axiom 2 $\forall q \varepsilon Q$ $\forall \sigma \varepsilon \underline{A}$ if $\ulcorner f(q,\sigma) \urcorner$ then $(\phi(q)-\{\langle\sigma\rangle\}) \subseteq \phi(f(q,\sigma))$.

It will be useful to define the auxiliary output function $g: Q \times \underline{A} \rightarrow 2^A$ by

$$\forall (q,\sigma) \varepsilon Q \times \underline{A} \quad g(q,\sigma)= \begin{cases} \phi(f(q,\sigma))-(\phi(q)-\{\langle\sigma\rangle\}) \text{ if } \ulcorner f(q,\sigma) \urcorner \\ \text{undefined otherwise} \end{cases}$$

Finally, B_o will always denote $\phi(q_o)$, the initial-operation set.

The set of natural numbers ω is intended to be the index set of a set of memory cells, each capable of being assigned some value from the universe of an interpretation. Each operation a may be thought of as a "black box" with corresponding input (domain) and output (range) connections to the memory cells.

A schema $S=(A,T)$ together with an interpretation I for A may be called a program (S,I). Under an interpretation, the operation a "computes" two functions, F_a and G_a. The function F_a is a "data-processing" function which performs a transformation on the memory, whereas G_a is a "decision" function which gives information about the memory to the transducer in the form of an element of the set $\underline{a}=\{a_1,a_2,....\}$. The transducer is responsible for assimilating this information, and based upon it, allows new operations to begin. Hence it may be thought of as a "sequential machine" which makes transitions as defined by f. For any state q, $\phi(q)$ is the set of operations which are enabled to be active, i.e. "computing", while the transducer is in state q. The auxiliary function g tells how ϕ is "updated" as new terminators occur. Hence, g rather than ϕ may be considered as the actual "output" of the transducer.

Axioms 1 and 2 are introduced to prevent an operation from being re-initiated while it is active. A transition from state q via a symbol σ

is defined if and only if the operation of which σ is a terminator can be active when T is in state q. Furthermore, no operation can be removed from $\phi(q)$ until it actually terminates. Parallelism in the model is possible because the transducer may allow several operations to be active concurrently; i.e. for some state q, $|\phi(q)| > 1$.

Figure 1 shows a simple example of a transducer. The customary use of labelled graphs is assumed[13]. The nodes (states) are labelled $q/\phi(q)$. The arcs are labelled with $\sigma\varepsilon\underline{A}$. The reader may wish to verify that the Axioms 1 and 2 are satisfied.

The initiation and termination of an operation will be called <u>events</u>. The occurrence of these events in time will be sufficient to completely describe all relevant activity in the model. By allowing an initiator to symbolize the initiation of an operation and allowing a terminator to symbolize the termination of an operation, activity within a schema may be represented by a sequence of initiators and terminators, providing it is assumed that at most one event can occur at a single instant. It will become obvious that this is not too stringent an assumption. Moreover, any disadvantages due to this assumption are outweighed by what is gained in tractability. Nothing will be assumed about either the time interval required for an operation to start once it is enabled, or for the time interval in which an operation is active.

The following describes the interaction of an operation a with the memory. For simplicity, it will be assumed that an operation a retrieves values from its domain D(a) only at the moment it initiates, and stores values in its range R(a) only at the moment it terminates. In the interim, there is no interaction with memory. Since the operation is to compute functions F_a and G_a, the particular set of values, one assigned to each domain cell, at the time of initiation will be designated by $\mu(a)$. A physical interpretation might be that $\mu(a)$ is a buffer. When an operation is not active, this buffer will be assigned the empty string, o. A formal definition of the behavior of a program is given below.

<u>Definition</u> Let $S=(A,T)$ be a schema and $I\varepsilon Int(A)$. <u>Conf(S,I)</u> denotes the set of <u>I-configurations</u> for S, i.e. quadruples of the form $\alpha=(q,B,d,\mu)$ where

(1) $q\varepsilon Q$
(2) $B\subseteq A$
(3) $d\varepsilon\nabla^{\omega}$
(4) μ assigns to each $a\varepsilon A$ either
  (i) o, the empty string
  or (ii) an element of $\nabla^{D(a)}$

More precisely, Conf(S,I) is defined in the following way: Define

$\alpha_o^I=(q_o,B_o,d_o,o^A)$ to be the <u>initial I-configuration</u>, where q_o is the initial transducer state, $B_o=\phi(q_o)$ is the initial operation set, d_o is the initial assignment, and o^A is that function μ such that $\forall a\varepsilon A \;\; \mu(a)=o$. Then Conf(S,I) is defined to be the smallest set containing α_o^I and closed under the set of partial functions $\{(\cdot\sigma) \mid \sigma\varepsilon\Sigma\}$ to be defined below.

Suppose $\alpha=(q,B,d,\mu)$. Then

(1) If $\sigma\varepsilon\bar{A}$, $\ulcorner\alpha\cdot\sigma\urcorner$ iff
  (i) $<\sigma>\varepsilon B$
  and (ii) $\mu(<\sigma>)=o$.
  In this case, $\alpha\cdot\sigma=(q,B-\{<\sigma>\},d,\mu')$ where

$$\forall a\varepsilon A \quad \mu'(a)=\begin{cases}\mu(a) \text{ if } a\neq<\sigma> \\ d|_{D(a)} \text{ if } a=<\sigma>\end{cases}$$

(2) If $\sigma\varepsilon\underline{A}$, $\ulcorner\alpha\cdot\sigma\urcorner$ iff
  (i) $<\sigma>\notin B$
  (ii) $\mu(<\sigma>)\neq o$
  and (iii) $\ulcorner f(q,\sigma)\urcorner$.
  In this case, $\alpha\cdot\sigma=(f(q,\sigma),B\bigcup g(q,\sigma),d',\mu')$ where

$$\forall m\varepsilon\omega \quad d'(m)=\begin{cases}d(m) \text{ if } m\notin R(<\sigma>) \\ F_{<\sigma>}\;(\mu(<\sigma>))(m) \text{ if } m\varepsilon R(<\sigma>)\end{cases}$$

$$\forall a\varepsilon A \quad \mu'(a)=\begin{cases}\mu(a) \text{ if } a\neq<\sigma> \\ o \text{ if } a=<\sigma>\end{cases}$$

The formal description above is meant to indicate how a given program (S,I) behaves. A single configuration $\alpha=(q,B,d,\mu)$ completely describes the relevant properties of (S,I) at a single instant. Those $\sigma\varepsilon\Sigma$ such that $\ulcorner\alpha\cdot\sigma\urcorner$ describe the possible transitions to a new configuration. The set B may be thought of as a "pool" of operations which may begin. The set $\{a \mid \mu(a)\neq o\}$ is the set of active operations. As defined above, an operation cannot be both active and in the pool simultaneously. The following algorithm, which is not necessarily deterministic or terminating, should complete the intuitive picture of how a program operates.

(1) Set $\alpha=\alpha_o^I=(q_o,B_o,d_o,o^A)$, i.e. the pool initially contains B_o, the state is q_o, the assignment is d_o, and $\mu(a)=o$ for every a.
(2) If for some $\sigma\varepsilon\Sigma$, $\ulcorner\alpha\cdot\sigma\urcorner$, fix σ and go to step (3). Otherwise stop.
(3) Replace α with $\alpha\cdot\sigma$. Go to step (2). The explanation of this step is in two cases. In the first case $\sigma=\bar{a}\varepsilon\bar{A}$. Then a is in the pool and $\mu(a)=o$. Remove a from the pool, put the values $d|_{D(a)}$ into the buffer $\mu(a)$ and go to step (2). In the second case $\sigma=a_j\varepsilon\underline{A}$. Change the state to $f(q,a_j)$, add $g(q,a_j)$ to the pool, replace the memory cells in R(a) with new values computed by $F_a(\mu(a))$, set $\mu(a)=o$, and go to step (2).

Since a transition between configurations α and α' may be represented by $\alpha\cdot\sigma=\alpha'$ for some $\sigma\varepsilon\Sigma$, it is appropriate to represent a sequence of transitions by sequences of elements of Σ. This is done formally in the next definition.

<u>Definition</u> Let $S=(A,T)$ be a schema, $I\varepsilon Int(A)$. For each $x\varepsilon\Sigma^*$ define a partial function $(\cdot x)$ mapping $Conf(S,I) \to Conf(S,I)$ inductively:

(1) $\forall\alpha\varepsilon Conf(S,I) \quad \alpha\cdot o=\alpha$
(2) $\forall x\varepsilon\Sigma^* \quad \forall\sigma\varepsilon\Sigma \quad \forall\alpha\varepsilon Conf(S,I) \quad \ulcorner\alpha\cdot(x\sigma)\urcorner$ iff

$\alpha\cdot x=\alpha'$ and $\ulcorner\alpha'\cdot\sigma\urcorner$, and then $\alpha\cdot(x\sigma)=(\alpha\cdot x)\cdot\sigma=\alpha'\cdot\sigma$.

Comp(S,I) is defined to be the subset of $\hat{\Sigma}$ of <u>I-computations</u> for S, where $x\varepsilon\hat{\Sigma}$ is in Comp(S,I) iff

(1) $\forall y \leq x \quad \ulcorner\alpha_o^I\cdot y\urcorner$
(2) If $x\varepsilon\Sigma^*$ then $\forall\sigma\varepsilon\Sigma$ not $\ulcorner\alpha_o^I\cdot x\sigma\urcorner$
(3) If $x\varepsilon\Sigma^\omega$ then $\forall\sigma\varepsilon\Sigma \quad \forall z < x \quad \exists y\varepsilon\Sigma^*$ $z \leq y < x$ and not $\ulcorner\alpha_o^I\cdot y\sigma\urcorner$

The reason for condition (1) is obvious. Condition (2) says that a computation terminates only if no further transitions can occur. Condition (3), called the <u>finite-delay property</u>[2], says that a possible transition (either the initiation or termination of an operation) cannot be delayed forever.

Define Pref(S,I)={y | y ≤ x, xεComp(S,I)}= the set of all prefixes of I-computations for S. Define Comp(S)= ⋃{Comp(S,I) | IεInt(A)}, and Pref(S)= ⋃{Pref(S,I) | IεInt(A)}.

<u>Example</u> Let A={a,b,c} be the operation set with $|\underline{a}|=|\underline{b}|=2$, $|c|=1$ and D and R given by the table below.

	D	R
a	{1}	{2}
b	{1}	{3}
c	{2,3}	{1}

Let S=(A,T) be the schema such that T is the transducer of Figure 1. Possible computations for two different interpretations I_1 and I_2 are as follows (It is left to the reader to invent the actual specifications for I_1 and I_2.):

For (S,I_1):

$$\bar{a}\ a_1\bar{b}\ b_1\bar{c}\ c_1$$

$$\bar{b}\ \bar{a}\ a_1 b_1\bar{c}\ c_1$$

$$\vdots$$

For (S,I_2):

$$\bar{a}\ a_1\bar{b}\ b_2\bar{a}\ a_1\bar{b}\ b_2\bar{a}\ a_1\bar{b}\ b_2\ldots$$

$$\bar{a}\ \bar{b}\ a_1 b_2\bar{a}\ \bar{b}\ b_2 a_1\bar{a}\ \bar{b}\ a_1 b_2\ldots$$

$$\vdots$$

However, the following is not a computation for any interpretation:

$$\bar{a}\ a_1\bar{b}\ b_2\bar{a}\ a_1\bar{b}\ b_1\bar{c}\ c_1$$

This is true because successive initiations of b must operate on the same values, since $R(a)\cap D(b)=\emptyset$, and hence the outcome b_2 followed by the outcome b_1 is inconsistent with the functionality of G_b for any interpretation. This phenomenon will be called a "repetition", precisely defined in the following.

<u>Definition</u> Let S=(A,T) be a schema, xεComp(S). Suppose $x=u\bar{a}v\bar{a}w$ for some aεA. The second indicated occurrence of $\bar{a}$ is called a <u>repetition</u> (of the first) if for every terminator σ occurring in v is $R(\langle\sigma\rangle)\cap D(a)=\emptyset$. A schema is called <u>repetition-free</u> if no repetitions occur in any computation.

The class of repetition-free schemata will play an important role in this development, because of the ease with which certain properties may be characterized for schemata in this class.

<u>Definition</u> A schema (A,T) is called <u>finite-state</u> if the state set of T is finite.

<u>Theorem 1</u> It is decidable whether a finite-state schema is repetition-free.

<u>1.2 Transducers</u>

The transducer of a schema has been defined in a form similar to the familiar Moore-type sequential machine, possibly with an infinite state set. Because the transition function f is partial, the machine is in a sense incomplete. However, in applying certain known automata-theoretic results, it will be treated as a complete machine by implicitly defining a new state to which every undefined transition must go. The next definition extends the available notation for describing transducers.

<u>Definition</u> Let (A,T) be a schema with $T=(Q,q_o,f,\phi)$. f is extended to a partial function f: $Q\times\underline{A}^* \rightarrow Q$ by the following induction:

(1) $\forall q\varepsilon Q \quad f(q,o)=q$
(2) $\forall q\varepsilon Q \quad \forall x\varepsilon\underline{A}^* \quad \forall\sigma\varepsilon\underline{A}$ if $f(q,x)=q'$ and $f(q',\sigma)=q''$ then $f(q,x\sigma)=q''$, otherwise $f(q,x\sigma)$ is undefined.

Also, define a partial function f_o: $\underline{A}^* \rightarrow Q$ by $\forall x\varepsilon\underline{A}^* \quad f_o(x)=f(q_o,x)$.
Define a partial function ϕ: $\underline{A}^* \rightarrow 2^A$ called the <u>behavior</u> of T (distinguishable from ϕ: $Q \rightarrow 2^A$ by context) to be

$$\forall x\varepsilon\underline{A}^* \quad \phi(x)=\begin{cases}\phi(f_o(x)) \text{ if } \ulcorner f_o(x)\urcorner \\ \text{undefined otherwise}\end{cases}$$

The transducer of a schema has been defined to have a countable, but not necessarily finite, set of states. To have physical significance it is usually desirable that infinite transducers have a "finite presentation". Examples of transducers of this sort appearing in the literature are Turing machines, pushdown machines, etc.

In applying any results of the aforementioned literature, two special properties are of interest. The first is known as the "on line" property. Interpreted in the present model, it means that the transducer produces one output $B\varepsilon 2^A$ in response to each input $\sigma\varepsilon\underline{A}$. This property is a consequence of the definition of a transducer already given. A second property is called "real time". A transducer has the real-time property if it is on line and if it changes its "internal configuration" at most once for each input. While this property is not necessary for transducers as they have been defined, it is a desirable one for investigation, since the ultimate goal is to speed up the overall execution of operations.

As an example of an infinite-state transducer having the real-time property, consider a "<u>real-time counter transducer</u>" which is a finite-state transducer augmented by a fixed number k of counters, each capable of recording a natural number. The transitions depend only on whether the counters have value 0 or not, and either increment, decrement, or leave unchanged any counter accordingly. A formal definition is omitted for brevity. It is shown in Section 4 that transducers of this type exhibit a rather gross inadequacy for controlling schemata in a maximally parallel fashion.

Transducers which employ counters are mentioned because much of the literature on parallel programs[2,4,5,6,14,15] concerns models with counters. The class of schemata with real-time counter transducers subsumes the class of "counter schemata"[2] and "flowgraph schemata"[15], if the properties of "persistence" and "permutability" are assumed for compatibility with the present model.

<u>Observation 1</u> One final form of a transducer which will have great utility later is the <u>labelled tree</u>. Although this tree may be infinite and not have a finite presentation, it is still well-defined. Let $S=(A,T)$, $T=(Q,q_o,f,\phi)$ be any schema. Define a transducer (Q',q_o',f',ϕ'), where Q' is to be the set of nodes of a tree and for $\sigma\varepsilon\underline{A}$ there is an arc (q,q') with label σ if $f(q,\sigma)=q'$. This can be done simply by letting $Q'=\{x\varepsilon\underline{A}^* \mid \ulcorner f_o(x)\urcorner\}$; $q_o' =o$ (the root of the tree); if $q=x\varepsilon\underline{A}^*$, then $f(q,\sigma)=x\sigma$, if $\ulcorner f_o(x\sigma)\urcorner$; and finally $\phi'(x)=\phi(f_o(x))$. Note that for the tree, the output function ϕ' and the behavior ϕ' are identical.

<u>1.3 Transduction Sets</u>

It is useful to have a description of allowable computations which is free of the internal workings of transducers and of references to interpretations. The following is a first step toward this goal.

<u>Definition</u> Let $S=(A,T)$ be a schema with $T=(Q,q_o,f,\phi)$. Define a partial function $\tau\colon Q\times 2^A\times 2^A\times\Sigma \rightarrow Q\times 2^A\times 2^A$ by $\forall(q,B,H)\varepsilon Q\times 2^A\times 2^A$

(1) $\forall\sigma\varepsilon\bar{A}$

$$\tau(q,B,H,\sigma)=\begin{cases}(q,B-\{<\sigma>\},H\bigcup\{<\sigma>\}) \text{ if} \\ \qquad <\sigma>\varepsilon B \text{ and } B\bigcap H=\emptyset \\ \\ \text{undefined otherwise}\end{cases}$$

(2) $\forall\sigma\varepsilon\underline{A}$

$$\tau(q,B,H,\sigma)=\begin{cases}(f(q,\sigma),B\bigcup g(q,\sigma),H-\{<\sigma>\}) \\ \qquad \text{if } <\sigma>\varepsilon H,\ B\bigcap H=\emptyset \text{ and} \\ \qquad \ulcorner f(q,\sigma)\urcorner \\ \\ \text{undefined otherwise}\end{cases}$$

τ is then extended to $\tau\colon Q\times 2^A\times 2^A\times\Sigma^* \rightarrow Q\times 2^A\times 2^A$ by $\forall(q,B,H)\varepsilon Q\times 2^A\times 2^A$

(1) $\tau(q,B,H,o)=(q,B,H)$

(2) $\forall x\varepsilon\Sigma^*$ $\forall\sigma\varepsilon\Sigma$ if $\tau(q,B,H,x)=(q',B',H')$ then

$$\tau(q,B,H,x\sigma)=\begin{cases}\tau(q',B',H',\sigma) \text{ if defined} \\ \\ \text{undefined otherwise}\end{cases}$$

Define the <u>transduction set</u> of T to be the set of $x\varepsilon\Sigma^*$ such that $\ulcorner\tau(q_o,B_o,\emptyset,x)\urcorner$. Denote this set by Σ^T.

An <u>allowed sequence</u> of T is an element of Σ^T. A <u>continuation</u> of an allowed sequence x is a string y such that xy is an allowed sequence.

<u>Observation 2</u> Upon comparing the behavior of τ with the definition of the transition between configurations of an interpreted schema, it will become clear that an allowed sequence is a string which could possibly be a prefix of a computation if the memory interconnections are ignored, i.e. it is a string satisfying the constraints which are due to the transducer only.

An element $(q,B,H)\varepsilon Q\times 2^A\times 2^A$ is intended to represent an instant in time when

(1) The transducer is in state q.

(2) $B\subseteq A$ is the set of operations waiting to initiate.

(3) $H\subseteq A$ is the set of operations which have initiated, but not terminated.

This connection will be made precise in Lemma 1. The function τ is closely related to the function τ of [2], if the states in the latter case are defined to be the set $Q\times 2^A\times 2^A$.

The following definitions provide useful terminology for describing transduction sets.

<u>Definition</u> For any $x\varepsilon\Sigma$, let $\underline{x}$ be the subsequence of x consisting of all and only those terminators in x, i.e. $\underline{x}=E(\underline{A},x)$. For any $P\subseteq\Sigma^*$ define $\underline{P}=\{\underline{x}\mid x\varepsilon P\}$, e.g. $\underline{\text{Pref}(S)}$, $\underline{\text{Comp}(S)}$.

Define $\tilde{\Sigma}$ to be those strings $x\varepsilon\Sigma^*$ such that for every $y \leq x$ and every $a\varepsilon A$, the number of initiators $\bar{a}$ in y minus the number of terminators of a in y is 0 or 1, i.e. $0 \leq (|E(\{\bar{a}\},y)|-|E(\underline{a},y)|) \leq 1$.

Define $\eta\colon \tilde{\Sigma} \rightarrow 2^A$ such that for any $x\varepsilon\tilde{\Sigma}$, $\eta(x)$ is the set of $a\varepsilon A$ such that the number of initiators $\bar{a}$ in x minus the number of terminators of a in x is exactly 1, i.e. $\eta(x)=\{a\varepsilon A \mid |E(\{\bar{a}\},x)|-|E(\underline{a},x)|=1\}$.

Hence if a sequence of events has occurred which is represented by x, $\eta(x)$ is the set of operations which have initiated but not terminated.

<u>Definition</u> Suppose $x\varepsilon\tilde{\Sigma}$ such that $x=u\bar{a}v$ for some $a\varepsilon A$, $u,v\varepsilon\Sigma^*$. Then an occurrence of an element σ of $\underline{a}$ in v, if one exists, is called the <u>mate</u> of $\bar{a}$ if $\bar{v}=v'\sigma v''$ and no $\pi\varepsilon v'$ is $\pi\varepsilon\underline{a}$. Similarly, the indicated occurrence of $\bar{a}$ is called the mate of σ.

Hence mates are pairs of symbols which correspond to a single activation of an operation.

<u>Lemma 1</u> Let T be a transducer and let τ be as defined above. Let $x\varepsilon\Sigma^*$. Then

(1) $x\varepsilon\Sigma^T$ iff

(i) $x\varepsilon\tilde{\Sigma}$

(ii) $\ulcorner f_o(\underline{x})\urcorner$

(iii) $\forall y \leq x \quad \eta(y) \subseteq \phi(\underline{y})$

and (2) if $x \varepsilon \Sigma^T$, letting

$$\tau(q_o, B_o, \emptyset, x) = (q', B', H')$$

(iv) $q' = f_o(\underline{x})$

(v) $H' = \eta(\overline{x})$

(vi) $H' \bigcup B' = \phi(\underline{x})$

The interpretation of (1) is that x is an allowed sequence if and only if (i) no terminators occur in x before corresponding initiators, (ii) $\underline{x}$ is a valid input to the transducer, and (iii) $\overline{\ }$the initiators occurring are only those which are allowed by the transducer. Properties (ii) and (iii) may be verified for a string x by inspecting the description of the transducer. The interpretation of (2) is exactly as promised in Observation 2. The proof involves a rather tedious induction on the length of x.

Theorem 2 Let A be an operation set, $P \subseteq \Sigma^*$, $P \neq \emptyset$. Then there exists a transducer T such that $P = \Sigma^T$ if and only if the following are true for all $x, y \varepsilon \Sigma^*$, $\sigma, \pi \varepsilon \Sigma$, and $a, b \varepsilon A$:

(1) $P \subseteq \tilde{\Sigma}$ (Unifold property)

(2) $x \varepsilon P$ and $y < x$ implies $y \varepsilon P$ (Prefix-closed property)

(3) $x\bar{a} \varepsilon P$ and $\sigma \varepsilon \underline{a}$ implies $x\bar{a}\sigma \varepsilon P$ (Immediate property)

(4) $x\sigma \varepsilon P$ and $x\pi \varepsilon P$ where $<\sigma> \neq <\pi>$ implies $x\sigma\pi \varepsilon P$ (Persistent property)

(5) $x\bar{a}\bar{b} \varepsilon P$ implies $x\bar{b} \varepsilon P$ (Permutable property)

(6) $x\bar{a}\pi\bar{b} \varepsilon P$ and $x\pi\bar{a} \varepsilon P$ implies $x\pi\bar{a}\bar{b} \varepsilon P$ (Semi-commutative property).

The proof is omitted.

Remark An explanation for the named properties is in order. The unifold property means that there must be one and only one initiation of each operation before each termination. (The name evolved from a more general model which allowed up to n copies of an operation to be active concurrently, this being called "n-fold".) The immediate property means that a terminator is allowed to occur immediately after its mate. The persistent property means that an event, once enabled for occurrence, cannot be disabled by an event from a different operation. The permutable property means that an initiation cannot be enabled solely by another initiation. The semi-commutative property prevents the order of occurrence between initiators and other symbols from affecting the occurrence of other initiators. It is related to the "commutative" property of [2]. The terms "persistent" and "permutable" are taken from [2].

1.4 Prefix Characterizations

This section states results which show that the prefixes of computations determine the computations themselves. The precise relation of transduction sets to prefixes is also determined.

Lemma 2 Let S=(A,T) be a schema, $I \varepsilon Int(A)$. Let $x \varepsilon \Sigma^*$. Then $x \varepsilon Pref(S,I)$ iff $\ulcorner \alpha_o^I \cdot x \urcorner$.

Definition For any operation set A, let S_o^A be a schema (A, T^o) such that $T^o = (\{q_o\}, q_o, f, \phi)$ where $\phi(q_o) = A$ and $\forall \sigma \varepsilon \underline{A} \quad f(q_o, \sigma) = q_o$.

Lemma 3 Let S=(A,T) be a schema, $I \varepsilon Int(A)$. Then $x \varepsilon Pref(S,I)$ iff $x \varepsilon Pref(S_o^I, I)$ and $x \varepsilon \Sigma^T$.

It can be noted that S_o^A allows any computation, as long as it is "consistent" with an interpretation. Hence the interpretation of this lemma is that the set of prefixes of computations of a schema S=(A,T) for an interpretation I is determined by two constraints:

(1) The transduction set Σ^T.

(2) The interpretation I.

Theorem 3 Let S=(A,T) be a repetition-free schema. Then $Pref(S) = \Sigma^T$.

Corollary 1 If S is repetition-free and $x, y \varepsilon Pref(S)$ such that $\tau(q_o, B_o, \emptyset, x) = \tau(q_o, B_o, \emptyset, y)$ then $\forall z \varepsilon \Sigma^* \quad xz \varepsilon Comp(S)$ iff $yz \varepsilon Comp(S)$.

Corollary 2 If S is repetition-free and $x, y \varepsilon Pref(S)$ such that $f_o(\underline{x}) = f_o(\underline{y})$ then $\forall z \varepsilon \underline{A}^*$ $\underline{x}z \varepsilon \underline{Comp(S)}$ iff $\underline{y}z \varepsilon \underline{Comp(S)}$.

The final result of this section demonstrates further the relation between computations and prefixes. It will be used in a rather tacit fashion in the sequel.

Theorem 4 Let $S_1 = (A, T_1)$, $S_2 = (A, T_2)$ be schemata. The following are equivalent:

(i) $Comp(S_1) = Comp(S_2)$

(ii) $Pref(S_1) = Pref(S_2)$

(iii) $\forall I \varepsilon Int(A) \quad Pref(S_1, I) = Pref(S_2, I)$

(iv) $\forall I \varepsilon Int(A) \quad Comp(S_1, I) = Comp(S_2, I)$

2. Determinacy and Equivalence

This section defines the notions of equivalence and determinacy of schemata. A syntactic characterization of these properties is presented for the class of repetition-free schemata.

2.1 Definitions

Let S=(A,T) be a schema, $I \varepsilon Int(A)$. For any $m \varepsilon \omega$, $x \varepsilon Comp(S,I)$ $\Omega_m^I(x)$ denotes the sequence of elements stored into cell m. (Note that the elements of $\Omega_m^I(x)$ are in one-to-one correspondence with terminators σ occurring in x such that $m \varepsilon R(<\sigma>)$.) S is called determinate if $\forall I \varepsilon Int(A)$ $\forall x, y \varepsilon Comp(S,I) \quad \forall m \varepsilon \omega \quad \Omega_m^I(x) = \Omega_m^I(y)$.

Let $S_1 = (A, T_1)$, $S_2 = (A, T_2)$ be schemata. S_1 and S_2 are called equivalent (written $S_1 \equiv S_2$) if $\forall I \varepsilon Int(A) \quad \{\Omega_m^I(x) | x \varepsilon Comp(S_1, I)\} = \{\Omega_m^I(y) | y \varepsilon Comp(S_2, I)\}$. S_1 and S_2 are called congruent (written $S_1 \cong S_2$) if $\forall I \varepsilon Int(A)$ $Comp(S_1, I) = Comp(S_2, I)$.

2.2 Serial Schemata

The following type of schema is determinate in a very trivial way.

Definition Let S=(A,T) be a schema, $I \varepsilon Int(A)$, $x \varepsilon Comp(S,I)$. Then x is said to be a serial computation if $\forall y < x$, letting $\alpha_o^I \cdot y = (q, B, d, \mu)$, there is at most one $a \varepsilon A$ such that $\mu(a) \neq o$.

A schema is called serial if every computation is serial.

A flowchart is a serial, finite-state schema. The alternate representation of a flowchart as shown in Figure 2 will be frequently used hereafter. This representation should be self-explanatory. Its validity is derived from the next lemma.

Lemma 4 Let S,I,x be as above. The following are equivalent:

(1) x is serial

(2) $\forall y<x \ |\phi(\underline{y})| \leq 1$

(3) $|B_0| \leq 1$ and $\forall y\sigma<x \ \ \sigma\varepsilon\underline{A}$ implies $|g(f_0(\underline{y}),\sigma)| \leq 1$

S is serial if and only if for each $I\varepsilon\mathrm{Int}(A)$, Comp(S,I) has exactly one element.
The proof is left to the reader.

Lemma 5 A schema is determinate if and only if it is equivalent to a serial schema.
Proof If a schema is equivalent to a serial schema, it is determinate by definition of the latter property and Lemma 4. Conversely, suppose $S=(A,T)$ is determinate, where $T=(Q,q_0,f,\phi)$. A second schema S' will be defined which has, for any interpretation, the same cell-sequence as S (there is only one for each $m\varepsilon\omega$) and which is serial. This is done simply by embedding the behavior of T in a second transducer T' which only allows one operation to become active at a time. Furthermore, the operations become active in a "round-robin" fashion, so that no operation is discriminated against for an arbitrarily long time, emulating the finite-delay requirement for S. Hence the computation for S' is one of the computations of S, and since S is determinate, $S \equiv S'$. Formally, T' is defined by $(Q\times P,(q_0,p_0),f',\phi')$ as follows: For any set $B\subseteq A$, let $\hat{B}$ be the operations of B listed in an arbitrary orde r, each operation appearing exactly once. P is defined to be the set of all such lists. p_0 is defined to be $\hat{B}_0$. For any $(q,p)\varepsilon Q\times P$, $\phi'(q,p)$ is defined to be the first component of the list p.

$$\forall(q,p)\varepsilon Q\times P \qquad f'((q,p),\sigma)=\begin{cases}(f(q,\sigma),p') \text{ if } \ulcorner f(q,\sigma)\urcorner\text{, where } p' \text{ is formed by deleting the first symbol of } p \text{ and concatenating } g(q,\sigma) \text{ to the opposite end of the list remaining.}\\ \text{undefined otherwise}\end{cases}$$

By the unifold property of S, no operation will ever appear twice in the list p.

Corollary 3 Every finite-state, determinate schema is equivalent to a flowchart.

Definition Let $x\varepsilon\tilde{\Sigma}$. Define the canonical sequence corresponding to x, denoted $\tilde{x}$, by the following induction:

(i) $\tilde{o}=o$

(ii) $$\widetilde{y\sigma}=\begin{cases}\tilde{y}<\sigma>\sigma & \text{if } \sigma\varepsilon\underline{A}\\ \tilde{y} & \text{if } \sigma\varepsilon\bar{A}\end{cases}$$

For example, if $x=\bar{a}\bar{b}\bar{c}a_1c_1\bar{c}c_1b_1\bar{c}\bar{d}$ then $\tilde{x}=\bar{a}a_1\bar{c}c_1\bar{c}c_1\bar{b}b_1$.

Corollary 4 Let S be a determinate schema. Let $x\varepsilon\mathrm{Pref}(S)$. Then there exists an equivalent serial schema S' such that $\tilde{x}\varepsilon\mathrm{Pref}(S')$.
Proof The construction is similar to that above, except that it must be guaranteed that the canonical sequence corresponding to x is the prefix of a computation. The details are left to the reader.

2.3 Characterizations of Determinacy

Presented here are results on determinacy of several different classes of schemata. The proofs, many of which are reminiscent of [2], are omitted.

Definition Let A be an operation set. Define a relation $\rho\subseteq A\times A$ by $\forall(a,b)\varepsilon A\times A \ \ (a,b)\varepsilon\rho$ iff at least one of the following is true:

(1) $R(a)\cap D(b)\neq\emptyset$.

(2) $R(b)\cap D(a)\neq\emptyset$.

(3) $a\neq b$ and $R(a)\cap R(b)\neq\emptyset$.

The notation $a\rho b$ means $(a,b)\varepsilon\rho$. $a\bar{\rho}b$ or $(a,b)\varepsilon\bar{\rho}$ means $(a,b)\varepsilon(A\times A)-\rho$. $(a,b)\varepsilon(\rho-I)$ means $(a,b)\varepsilon\rho$ and $a\neq b$. (Note: "I" should not be confused with an interpretation in this case.) If $(a,b)\varepsilon(\rho-I)$, a and b may be said to conflict.

Definition A schema $S=(A,T)$ is called conflict-free if $\forall x\varepsilon\mathrm{Pref}(S) \ \ \forall(a,b)\varepsilon(\rho-I) \ \ \{a,b\}\not\subseteq\phi(\underline{x})$.

Definition A schema $S=(A,T)$, $T=(Q,q_0,f,\phi)$ is called commutative if $\forall q\varepsilon Q \ \ \forall\sigma,\pi\varepsilon\underline{A}$ if $<\sigma>\neq<\pi>$ and $\{<\sigma>,<\pi>\}\subseteq\phi(q)$ then $f(q,\sigma\pi)=f(q,\sigma\pi)$.

Definition Let A be an operation set. For $x,y\varepsilon\tilde{\Sigma}$ define $x \sim y$ to hold iff

(1) $\forall a\varepsilon A \ \ E(\{\bar{a}\},x)=E(\{\bar{a}\},y)$ and

(2) $\forall(a,b)\varepsilon\rho \ \ E(\{\bar{a},\bar{b}\},x)=E(\{\bar{a},\bar{b}\},y)$

Let $S=(A,T)$ be a schema. S is said to be syntactically determinate if $\forall I\varepsilon\mathrm{Int}(A)$ $\forall x,y\varepsilon\mathrm{Comp}(S,I) \ \ x \sim y$.

Table I summarizes the known characterizations of determinacy[1].

2.4 Characterization of Equivalence

This section presents, without proof, a syntactic characterization of equivalence for repetition-free schemata.

Lemma 6 Let $S_1=(A,T_1)$, $S_2=(A,T_2)$. If $S_1 \equiv S_2$ then S_1 and S_2 are either both repetition-free or neither is.

Definition Schemata $S_1=(a,T_1)$ and $S_2=(A,T_2)$ are said to be syntactically equivalent iff $\forall I\varepsilon\mathrm{Int}(A)$

(1) $\forall a\varepsilon A \ \ \{E(\{\bar{a}\},x)|x\varepsilon\mathrm{Comp}(S_1,I)\}=\{E(\{\bar{a}\},y)|y\varepsilon\mathrm{Comp}(S_2,I)\}$ and

(2) $\forall(a,b)\varepsilon\rho \ \ \{E(\{\bar{a},\bar{b}\},x)|x\varepsilon\mathrm{Comp}(S_1,I)\}=\{E(\{\bar{a},\bar{b}\},y)|y\varepsilon\mathrm{Comp}(S_2,I)\}$

Theorem 5 Let $S_1=(A,T_1)$, $S_2=(A,T_2)$ be determinate, repetition-free schemata. Then $S_1 \equiv S_2$ if and only if S_1 and S_2 are syntactically equivalent.

The flowcharts of Figure 3 show that the repetition-free hypothesis cannot be eliminated from the preceding theorem.

3. Defining Maximal Parallelism

Formal definitions will first be given which describe the relative parallelism between schemata. These definitions will then be shown to be in harmony with an alternate characterization incorporating "time assignments".

Definition Let $S_1=(A,T_1)$, $S_2=(A,T_2)$ be equivalent schemata. Define a relation $S_1 \leq S_2$ to hold if $\forall I\varepsilon Int(A)$ $Comp(S_1,I)\subseteq Comp(S_2,I)$. Define $S_1 < S_2$ to hold if $S_1 \leq S_2$ and $\exists I\varepsilon Int(A)$ $Comp(S_1,I)\neq Comp(S_2,I)$.

A schema $S_1=(A,T_1)$ is called closed if $\forall S_2=(A,T_2)$ $S_2 \equiv S_1$ implies $S_2 \leq S_1$.

A schema $S_1=(A,T_1)$ is called a closure of $S_2=(A,T_2)$ if S_1 is closed and $S_1 \equiv S_2$.

$S_1 < S_2$ means S_2 is "more parallel" than S_1. A schema is closed if it is "maximally parallel" among all equivalent schema. A schema S_2 is a closure of S_1 if it is a "maximally parallel equivalent" of S_1.

The proofs of the following are left to the reader.

Lemma 7 Let $S=(A,T)$ be a schema. S is closed if and only if $\forall I\varepsilon Int(A)$ $Comp(S,I)= \bigcup_{S'\equiv S} Comp(S',I)$.

Lemma 8 Closures of equivalent schemata are congruent, i.e. they have exactly the same computations for each interpretation.

The previous definitions will now be justified by an alternate relation on schemata based on timing. Certain limitations, such as determinacy, etc. will be applied for brevity. This argument is not essential for an understanding of the rest of the paper.

Definition V denotes the set of non-negative real numbers. V^+ denotes $V-\{0\}$.

Let A be an operation set. A time assignment for A is a set $\tau=\{\tau^a|a\varepsilon A\}$ where $\forall a\varepsilon A$ $\tau^a\colon \omega \rightarrow V$ and $\exists M\varepsilon V^+$ $\forall a\varepsilon A$ $\forall i\varepsilon\omega$ $\tau^a(i) \geq M$. TA(A) denotes the set of all time assignments for A.

The meaning of τ will be as follows: If there is an i-th activation of operation a in a computation then $\tau^a(i)$ is the time interval of this activation, i.e. the time elapsed between the occurrence of $\bar{a}$ and its mate. The number M is a positive lower bound on all time intervals.

Definition Let $S=(A,T)$ be a schema, $I\varepsilon Int(A)$, $\tau\varepsilon TA(A)$. Define a function $t\colon Pref(S,I) \rightarrow V$ inductively as follows:

(i) $t(o)=0$
(ii) $t(x\bar{a})=t(x)$
(iii) If $\sigma\varepsilon \underline{a}$ then $t(x\sigma)=t(u)+\tau^a(i)$, where u and i are such that $x=u\bar{a}v$ with $\bar{a}$ the mate of σ, and i is the number of occurrences of $\bar{a}$ in x.

If S, I, and τ are as above, a string $x\varepsilon Pref(S,I)$ $Comp(S,I)$ is said to be consistent with τ if $t({}_1x) \leq t({}_2x) \leq t({}_3x) \leq \dots$.

Remark The interpretation of t(x) is obviously the sequence of times at which events occur in x. For simplicity, previous assumptions are modified slightly to allow some events to occur simultaneously. Also, for any τ there is always at least one consistent computation, e.g. any serial computation.

Definition Let S, I, and τ be as above, and suppose $z\varepsilon Pref(S,I)\bigcup Comp(S,I)$ is consistent with τ. Define the timing of z (relative to τ) to be the set $s_z^a=\{s_z|a\varepsilon A\}$, where $\forall a\varepsilon A$ $s_z^a\colon \omega \rightarrow V^+$ is defined by

$$s_z^a(i)=\begin{cases} t(u\sigma) \text{ where } u\sigma \leq z \text{ is such that } \sigma \text{ is} \\ \quad \text{the i-th terminator of a in z,} \\ \quad \text{if any} \\ \\ \text{undefined otherwise} \end{cases}$$

Hence if there is an i-th occurrence of a in z, $s_z^a(i)$ is the time at which this occurrence teminates and stores its range values.

Definition Let $S_1=(A,T_1)$, $S_2=(A,T_2)$ be equivalent, determinate schemata. Let $I\varepsilon Int(A)$, $\tau\varepsilon TA(A)$, $z\varepsilon Comp(S_1,I)$, and $z'\varepsilon Comp(S_2,I)$ such that z and z' are consistent with τ. For any $a\varepsilon A$ define $s_z^a \leq s_{z'}^a$ iff $\forall i\varepsilon\omega$ $s_z^a(i) \leq s_{z'}^a(i)$. Define

$s_z^a < s_{z'}^a$ iff $s_z^a \leq s_{z'}^a$ and $\exists i\varepsilon\omega$ $s_z^a(i) < s_{z'}^a(i)$.

Define $s_z \leq s_{z'}$ (z is as fast as z', with respect

to τ) iff $\forall a\varepsilon A$ $s_z^a \leq s_{z'}^a$. Define $s_z < s_{z'}$ (z is

faster than z', with respect to τ) iff $s_z^a \leq s_{z'}^a$ and

$\exists a\varepsilon A$ $s_z^a < s_{z'}^a$.

Definition Let $S_1=(A,T_1)$, $S_2=(A,T_2)$ be equivalent, determinate schemata. Define $S_1 \preceq S_2$ (S_2 is as fast as S_1) iff $\forall I\varepsilon Int(A)$ $\forall \tau\varepsilon TA(A)$ $\forall z\varepsilon Comp(S_1,I)$ if z is consistent with τ then $\exists z'\varepsilon Comp(S_2,I)$ $s_{z'} \leq s_z$. Define $S_1 \prec S_2$ (S_2 is faster than S_1) iff $S_1 \preceq S_2$ and $\exists I\varepsilon Int(A)$ $\exists \tau\varepsilon TA(A)$ $\exists z\varepsilon Comp(S_2,I)$ such that z is consistent with τ and $\forall \xi\varepsilon Comp(S_1,I)$ if ξ is consistent with τ then $s_z < s_\xi$.

The following clearly demonstrates the duality between timing and parallelism.

Theorem 6 Let $S_1=(A,T_1)$, $S_2=(A,T_2)$ be equivalent, commutative, repetition-free, determinate schemata. Then $S_1 \leq S_2$ iff $S_1 \preceq S_2$, and $S_1 < S_2$ iff $S_1 \prec S_2$.

Presented in the following are several alternate characterizations of maximal parallelism. Incorporated in this presentation is a series of results which show how the parallelism of a schema can be increased by a "look-ahead" procedure.

Definition Let $S=(A,T)$ be a schema. Define a predicate Ult_S on $A \times \underline{\text{Pref}(S)}$ by

$$\forall b \varepsilon A \quad \forall x \varepsilon \text{Pref}(S) \quad \text{Ult}_S(b,\underline{x}) \text{ iff}$$

$$\forall z \varepsilon \text{Comp}(S) \text{ if } x < z \text{ then} \exists y \varepsilon \Sigma^* \quad xy < z \text{ and}$$

$$b \varepsilon \phi(\underline{xy}) \text{ and } \forall u < y \quad \forall a \varepsilon \phi(\underline{xu}) \quad b \bar{\rho} a.$$

The subscript "S" may be omitted when S is understood.

Informally, in view of the finite-delay property, Ult$(b,\underline{x})$ is true iff for every computation z with prefix either (i) b occurs in x unmated; or (ii) b will ultimately occur in z after x and no conflicting operations will become active in the meantime; and in both cases no conflicting operations are concurrently active with b. For Ult to be well-defined, the definition must be shown to be dependent only on $\underline{x}$, rather than x. It is simple to show that, if $x,x' \varepsilon$Pref (S) are such that $\underline{x} = \underline{x}'$ then for any $I \varepsilon$Int(A), $\alpha_o^I \circ x$ and $\alpha_o^I \circ x'$ differ only with regard to which operations in $\phi(\underline{x})$ have initiated. It follows that if either (i) or (ii) holds for x, then (i) or (ii) holds for x'.

If $b \varepsilon \phi(\underline{x})$ then Ult$(b,\underline{x})$. The importance of this predicate is that if Ult$(b,\underline{x})$ but $b \notin \phi(\underline{x})$, then it will be possible to modify the schema S so that b is enabled in state $f_o(\underline{x})$, without having the effect of changing the cell sequences.

Definition A schema $S=(A,T)$ is called globally complete if $\forall x \varepsilon \text{Pref}(S) \quad \forall b \varepsilon A \quad \text{Ult}(b,\underline{x})$ implies $b \varepsilon \phi(\underline{x})$.

Hence a schema is globally complete if no operations can be enabled by looking ahead, other than those already enabled.

The following is the first part of a syntactic characterization of closed schemata.

Theorem 7 Every closed schema is globally complete.

Proof Suppose $S=(A,T)$ is not globally complete. Let $x \varepsilon$Pref(S), $b \varepsilon A$ be such that Ult$(b,\underline{x})$ and $b \notin \phi(\underline{x})$. A new schema $\hat{S}=(A,\hat{T})$ will be constructed such that $\hat{S} \equiv S$ and $\hat{S} < S$, thereby showing that S is not closed. By Observation 1, it may be assumed without loss of generality that T is a tree. From S, the construction of $\hat{S}$ proceeds according to

Construction 1 Let $T=(Q,q_o,f,\phi)$. Define $q_x = f_o(\underline{x})$. Q will be decomposed into $Q^1 \bigcup Q^2 \bigcup Q^3$, where each pair of subsets is disjoint. Q^1 and Q^2 are defined inductively by the following rules:

Q^1:

(i) $q_x \varepsilon Q^1$

(ii) If $q \varepsilon Q^1$and $\exists \sigma \varepsilon \underline{A} \quad q'=f(q,\sigma)$ and $b \notin \phi(q')$ then $q' \varepsilon Q^1$

(iii) The only elements of Q^1 are those obtained by a finite number of applications of the above rules.

Q^2:

(i) If $q \varepsilon Q^1$ and $\exists \sigma \varepsilon \underline{A} \quad q'=f(q,\sigma)$ and $b \varepsilon \phi(q')$, then $q' \varepsilon Q^2$.

(ii) If $q \varepsilon Q^2$ and $\exists \sigma \varepsilon (\underline{A}-\underline{b}) \quad q'=f(q,\sigma)$, then $q' \varepsilon Q^2$.

(iii) The only elements of Q^2 are those obtained by a finite number of applications of the above rules.

$Q^3 = Q-(Q^1 \bigcup Q^2)$.

In summary, Q^1 is the set of states reachable from q_X by a sequence $y \varepsilon \underline{A}^*$ such that $b \notin \phi(\underline{xy})$ but Ult$(b,\underline{xy})$. Q^2 is the set of states reachable from states in Q^1 for which the ultimate b actually becomes enabled.

Under the assumption that T is a tree, Q^1, Q^2 and Q^3 are clearly disjoint, since if $y \neq y'$ then $f(q,y) \neq f(q,y')$.

The plan now is to construct a schema $\hat{S}$ which behaves as S, except that $\hat{S}$ enables b when in state q_X. To do so, account must be taken for the possibility that b terminates in $\hat{S}$ prior to the point at which it would terminate in S. The corresponding terminator of b must be "recorded" in additional states of $\hat{S}$, until the point is reached in schema S where a transition with this terminator would normally be defined. Thus for each $q \varepsilon Q^2$ and each $\pi \varepsilon \underline{b}$, the termination of b will be recorded in a new state named q^π. Define $Q^4 = \{q^\pi | q \varepsilon Q^2, \ \pi \varepsilon \underline{b}\}$. Define $\hat{T}=(\hat{Q},q_o,\hat{f},\hat{\phi})$, where $\hat{Q}=Q^4 \bigcup Q$ and $\hat{\phi}$and $\hat{f}$ are given in Table II. A formal proof of the equivalence of $\hat{S}$ and S is omitted. This construction is demonstrated in Figures 4 and 5.

For the repetition-free case, every allowed sequence is the prefix of a computation. This fact provides for the use of Construction 1 on repetition-free schemata of arbitrary (not necessarily tree) structure. This will be shown by the lemmas to follow.

Lemma 9 If $\text{Ult}_S(b,\underline{x})$ and y is such that $x < y$, $y \varepsilon$Pref(S), and no terminator of b occurs in y, then $\text{Ult}_S(b,\underline{y})$. The proof is an immediate consequence of the definition of Ult.

Lemma 10 If S is repetition-free, the value of $\text{Ult}_S(b,\underline{x})$ for a given b depends only on $f_o(\underline{x})$, i.e. $f_o(\underline{x})=f_o(\underline{y})$ implies Ult$(b,\underline{x})$ iff Ult$(b,\underline{y})$. The proof follows from the definition of Ult and Corollary 4

Definition Let $S=(A,T)$ be a repetition-free schema, q a state of T, $b \varepsilon A$. Define $\text{Ult}_S(b,q)$ to be true iff $\exists x \varepsilon \text{Pref}(S) \quad f_o(\underline{x})=q$ and $\text{Ult}_S(b,\underline{x})$.

Corollary 5 (Lemma 10) If S is repetition-free, $\text{Ult}_S(b,q)$ iff $\forall x \varepsilon \text{Pref}(S) \quad f_o(\underline{x})=q$ implies $\text{Ult}_S(b,\underline{x})$.

Lemma 11 Let $S=(A,T)$ be any repetition-free, determinate schema which is not globally complete. Let $x \varepsilon \mathrm{Pref}(S)$, $b \varepsilon A$ be such that $\mathrm{Ult}(b,\underline{x})$ and $b \not\varepsilon \phi(\underline{x})$. The result of applying Construction 1 to S yields $\hat{S}$, an equivalent schema such that $S < \hat{S}$. Moreover if S is commutative, so is $\hat{S}$. If S is finite-state, so is $\hat{S}$.

Proof It is necessary to show that for the set Q^1 of the construction, any continuation from a particular $q \varepsilon Q^1$ is independent of how q was reached. This follows from Corollary 1. (This is not generally true in the non-repetition-free case. This is why the tree representation was assumed in Theorem 7, for then the statement is true.) From Lemma 9 and Corollary 5, it follows that if $q \varepsilon Q^1$ then $\mathrm{Ult}_S(b,q)$.

To show that commutativity is preserved, it is necessary to enumerate the different cases in the definition of $\hat{f}$. Only one case will be demonstrated here. Suppose $q \varepsilon Q^1$, $\pi \varepsilon \underline{b}$, σ an arbitrary element of $\underline{A}-\underline{b}$. Assume $\ulcorner f(q,\sigma\pi)\urcorner$ and $\ulcorner f(q,\pi\sigma)\urcorner$. Then $\hat{f}(q,\pi\sigma) = f(q^{\pi},\sigma) =$

$$\left.\begin{cases} f(q,\sigma)^{\pi} \text{ if } f(q,\sigma)\varepsilon Q^1 \\ \\ f(q,\sigma\pi) \text{ if } f(q,\sigma)\varepsilon Q^2 \end{cases}\right\} = \hat{f}(q,\sigma\pi)$$

Finiteness is clearly preserved since only finitely many new states are added if Q is finite. Figure 6 shows the construction for this special case.

One final result will prove sufficient for characterizing closed, repetition-free schemata.

Lemma 12 Let S, S' be equivalent, determinate, repetition-free schemata. If $x \varepsilon \mathrm{Pref}(S) \cap \mathrm{Pref}(S')$ and $b \varepsilon A$ such that $\mathrm{Ult}_S(b,\underline{x})$, then $\mathrm{Ult}_{S'}(b,\underline{x})$. The proof follows from Theorem 5 and the definition of Ult.

Theorem 8 Let S be a determinate, repetition-free schema. S is closed if and only if S is globally complete.

Proof S closed implies S globally complete is Theorem 7. Suppose $S=(A,T)$ is a determinate, repetition-free schema and S is not closed. Let $S'=(A,T')$ be such that $S' \equiv S$ and for some $I \varepsilon \mathrm{Int}(A)$ $\mathrm{Comp}(S',I) \not\subseteq \mathrm{Comp}(S,I)$. Let $x \varepsilon \mathrm{Pref}(S')-\mathrm{Pref}(S)$ such that x is minimal with respect to $\leq$. Then $x=y\sigma$ where $y \varepsilon \mathrm{Pref}(S') \cap \mathrm{Pref}(S)$ and $\sigma \varepsilon \Sigma$. σ is an initiator, since if σ were a terminator, then $y\sigma \varepsilon \mathrm{Pref}(S') \cap \mathrm{Pref}(S)$, a contradiction. Let $\sigma=\bar{b}$. Then $b \not\varepsilon \phi(\underline{x})$, but $b \varepsilon \phi'(\underline{x})$, where ϕ, ϕ' are the behaviors of S, S' respectively. Then $\mathrm{Ult}_{S'}(b,\underline{x})$, and by Lemma 12, $\mathrm{Ult}_S(b,\underline{x})$; therefore S is not globally complete.

The schema S of Figure 3 shows that the repetition-free hypothesis cannot be eliminated, since this schema is not closed, but is globally complete.

Definition A schema $S=(A,T)$ is called *locally complete* if $\forall x \varepsilon \mathrm{Pref}(S)$ $\forall (a,b)\varepsilon\bar{\rho}$ if $a \varepsilon \phi(\underline{x})$ and $\forall \sigma \varepsilon \underline{a}$ $b \varepsilon \phi(\underline{x\sigma})$ then $b \varepsilon \phi(\underline{x})$.
(Local completeness is analogous to being in "maximum parallel form" in [15].)

Theorem 9 Let $S=(A,T)$ be a repetition-free, determinate schema. S is globally complete if and only if S is locally complete.

Proof Suppose S is not locally complete. Let $x \varepsilon \mathrm{Pref}(S)$, $(a,b)\varepsilon\,\bar{\rho}$ be such that $a \varepsilon \phi(\underline{x})$ and $\forall \sigma \varepsilon \underline{a}$ $b \varepsilon \phi(x\sigma)$ but $b \not\varepsilon \phi(\underline{x})$. Clearly $\mathrm{Ult}(b,\underline{x})$ since $\bar{b}$ will always occur sometime after the termination of a, by Theorem 5. Since $b \not\varepsilon \phi(\underline{x})$, S is not globally complete. Conversely, suppose S is not globally complete, but S is locally complete. Let $x \varepsilon \mathrm{Pref}(S)$, $b \varepsilon A$ such that $\mathrm{Ult}_S(b,\underline{x})$ and $b \not\varepsilon \phi(\underline{x})$. Assume, without loss of generality, that T is a tree, by Observation 1. Let Ψ be the subtree of T rooted at $\underline{x}$ and truncated at any path u such that $b \varepsilon \phi(\underline{xu})$.

Claim $\exists v \varepsilon \underline{A}^*$ $\exists a \varepsilon A$ $\forall \pi \varepsilon \underline{a}$ $v\pi$ is a maximal path in Ψ. This claim is the same is saying that $\exists v \varepsilon \underline{A}^*$ $\exists a \varepsilon A$ $a\bar{\rho}b$ and $\forall \pi \varepsilon \underline{a}$ $b \varepsilon \phi(\underline{xv\pi})$, by the definition of Ψ. Since $b \not\varepsilon \phi(\underline{xv})$, this would contradict the assumption that S is locally complete. If the tree Ψ is finite then the claim is obvious. Suppose that Ψ is infinite and the claim is false. Then for any finite path v from the root of Ψ, for all $c \varepsilon \phi(\underline{xv})$ there must be at least one $\gamma \varepsilon \underline{c}$ such that $b \not\varepsilon \phi(\underline{xv\gamma})$, otherwise the claim would be true. In the manner of constructing an infinite path in the proof of König's Lemma[16], an infinite path w in Ψ may be constructed such that $\forall w' < w$ $b \not\varepsilon \phi(\underline{xw'})$. By judiciously choosing the operation c at each step of the construction, i.e. in the "round robin" fashion (see Lemma 5), the canonical sequence corresponding to path $\underline{xw}$ satisfies the finite-delay property and therefore is a computation. ("Canonical sequence" was defined only for finite strings. It is extended to infinite strings in the obvious way.) By construction, $\bar{b}$ does not occur in this computation after x, therefore not $\mathrm{Ult}_S(b,\underline{x})$ contradicting the hypothesis.

Corollary 6 A repetition-free, determinate schema is closed if and only if it is locally complete.

An application of this Corollary is

Theorem 10 It is decidable whether a repetition-free, determinate, finite-state schema is closed.

Proof Such a schema S is closed iff it is locally complete, by Corollary 6. S is not locally complete iff for some reachable state q and some $a \varepsilon \phi(q)$ $\exists b \not\varepsilon \phi(q)$ such that $b\bar{\rho}a$ and $\forall \sigma \varepsilon \underline{a}$ $b \varepsilon \phi(f(q,\sigma))$. This is clearly decidable.

Theorem 11 Let S be a determinate, repetition-free schema. Then a closure of S exists. In fact a commutative closure of S exists.

Before presenting the proof, it will be useful to agree on a convention for Construction 1 which makes it a single-valued transformation on schemata.

Convention Let S be a non-closed, repetition-free schema. Unless otherwise specified, the notation $\hat{S}$ will be agreed to mean the result of applying Construction 1 to S, such that

(1) x is a least length prefix such that for some $b \varepsilon A$ $\mathrm{Ult}_S(b,\underline{x})$ and $b \not\varepsilon \phi(\underline{x})$

(2) If there is a choice of b or x in (1), then they are chosen by some fixed ordering of A and Σ respectively.

Proof of Theorem 11 Let S be a repetition-free, determinate schema. Without loss of generality, by Lemmas 8 and 10, it suffices to let S be a serial schema and therefore S is commutative. Define an infinite sequence of schemata $S_0,S_1,S_2,\ldots$ by letting $S_0=S$ and, having defined S_i, define

$$S_{i+1}=\begin{cases} S_i \text{ if } S_i \text{ is closed} \\ \hat{S}_i \text{ otherwise}\end{cases}$$

For $i\varepsilon\omega$ let $P_i=\text{Pref}(S_i)$ and ϕ_i be the behavior of S_i. By Lemma 4, $\forall i\varepsilon\omega \;\; P_i\subseteq P_{i+1}$, and S_i is repetition-free, determinate, commutative, and equivalent to S. Define $P=\bigcup_{i\varepsilon\omega} P_i$. It is now claimed that

(i) P is the transduction set of a commutative schema $\tilde{S}$.
(ii) $\tilde{S}$ is repetition-free.
(iii) $\tilde{S}$ is determinate.
(iv) $\tilde{S}\equiv S$.
(v) $\tilde{S}$ is closed.

To show (i) it suffices to show that P satisfies the conditions (1)-(6) of Theorem 2 and commutativity. Since each $P_i\subseteq\tilde{\Sigma}$, also $P\subseteq\tilde{\Sigma}$ and (1) is satisfied. If $x\varepsilon P$ then for some i, $x\varepsilon P_i$, and $\forall y<x \;\; y\varepsilon P_i$, therefore $y\varepsilon P$ and (2) is satisfied. The proofs of (3)-(6) and commutativity are similar. To show (ii) that S is repetition-free, suppose there is a computation of S with a repetition. Then there is a prefix of S with a repetition, hence a prefix of some S_i with a repetition. However S_i is repetition-free, so this is a contradiction. To show (iii) it will be shown that $\tilde{S}$ is conflict-free, determinacy following by the results of Section 2. Suppose $\exists x\varepsilon\text{Pref}(S)$ $\exists(a,b)\varepsilon(\rho-I)$ and $\{a,b\}\subseteq\tilde{\phi}(\underline{x})$. Then for some S_i, $\{a,b\}\subseteq\phi_i(\underline{x})$, contradicting the determinacy of S_i. For (iv) assume $S\not\equiv S$. Then $\exists I\varepsilon\text{Int}(A)$ $\exists x\varepsilon\text{Comp}(S,I)$ $\exists y\varepsilon\text{Comp}(S,I)$ $\exists m\varepsilon\omega$ $\Omega_m(x)\neq\Omega_m(y)$. Since every prefix of y is in $\text{Pref}(\tilde{S})$, also $y\varepsilon\text{Comp}(\tilde{S},I)$. However this is contrary to the determinacy of $\tilde{S}$. Finally it must be shown that (v) $\tilde{S}$ is closed. Suppose not. Then $\tilde{S}$ is not globally complete, i.e. $\exists x\varepsilon\text{Pref}(S)$ $\exists b\varepsilon A$ $\text{Ult}_{\tilde{S}}(b,\underline{x})$ and $b\notin\tilde{\phi}(\underline{x})$. Fixing x and b, let $i\varepsilon\omega$ be such that $x\varepsilon P_i$. Since there are only a finite number of strings of length less that or equal to x, for some $j\geq i$, x is the minimal prefix such that $\text{Ult}_{S_j}(b,\underline{x})$ and $b\notin\phi_j(\underline{x})$. Then by definition of S_{j+1}, $b\varepsilon\phi_{j+1}(\underline{x})$, hence $b\varepsilon\tilde{\phi}(\underline{x})$, a contradiction.

To see that the repetition-free hypothesis cannot be eliminated, observe S_1 of Figure 5. By enumerating computations, the only other equivalent schema is S_2, hence the set of computations of a closure must be $\{\bar{a}a_1\bar{b}b_1\bar{c}c_1\bar{a}a_1,\ \bar{a}a_1\bar{c}c_1\bar{a}a_1\bar{b}b_1\}$. However there is no schema with only these computations. Hence no closure of S_1 exists.

4. Complexity of Closures

The previous section demonstrated the existence of a closure for any repetition-free determinate schema, regardless of the complexity of the transducer. In this section, the complexity of closures of the finite-state schemata will be investigated. The emphasis will be on repetition free, determinate schemata, since several helpful characterizations of closed schemata for this class are available.

Theorem 12 Let S be a determinate, repetition-free, finite-state schema. The predicate Ult_S is recursive, i.e. it is decidable given $x\varepsilon\text{Pref}(S)$, $b\varepsilon A$, whether $\text{Ult}_S(b,\underline{x})$.
Proof By Corollary 4 and Lemma 12, it is sufficient to prove this for the case that S is a flowchart. This is done by

Construction 2 Let $S=(A,T)$, $T=(Q,q_0,f,\phi)$, be a repetition-free flowchart. Let $x\varepsilon\text{Pref}(S)$, $b\varepsilon A$, and $q_x=f_0(\underline{x})$. Construct a tree Ψ with nodes labelled from Q with q_x as the root. At each stage of the construction the nodes are in one of three classes: neutral, frontier, and end. The following algorithm is that for the construction of Ψ.

1. q_x is the initial frontier node and the root of the tree.
2. If there is no frontier node then stop. Otherwise let n be such a node and let q be its label.
3. If there is a predecessor of n with label q, then n becomes an end node. In this case go to step 2. Otherwise continue.
4. If there is a σ such that $\ulcorner f(q,\sigma)\urcorner$, go to step 5. Otherwise n becomes an end node. Go to step 2.
5. For each σ such that $\ulcorner f(q,\sigma)\urcorner$, add to the tree a new son of n labelled with the state $f(q,\sigma)$, which becomes a frontier node. n now becomes a neutral node. Go to step 2.

Figure 7 shows Ψ for a particular flowchart and a particular x after the construction is complete. Observe that the construction has duplicated the behavior of the transducer until either a cycle occurs, or until a state q is reached for which $\phi(q)=\emptyset$, hence the construction always terminates. It is claimed that $\text{Ult}(b,\underline{x})$ can be computed for any b from Ψ as follows.
Claim $\text{Ult}(b,\underline{x})$ if and only if (*) on every path from the root of Ψ there is a node with label q such that $b\varepsilon\phi(q)$ and for no predecessor n' of n with label q' is there an $a\varepsilon\phi(q')$ such that $a\rho b$.
Proof Suppose (*) holds. Then clearly every computation with prefix x is such that $\bar{b}$ occurs after x (or $\bar{b}$ is unmated in x), and moreover, no operation which is ρ-related to b can occur prior to $\bar{b}$. Formally stated, this is precisely the case that $\text{Ult}(b,\underline{x})$. Conversely, suppose (*) fails. Then there is a path with no node labelled q such that $b\varepsilon\phi(q)$, or for some path with such a node, an operation ρ-related to b occurs prior to the first occurrence of $\bar{b}$. In the latter case it is clear that not $\text{Ult}(b,\underline{x})$. In the former case there are two subcases. If the last node of the path has label q where $\phi(q)=\emptyset$, then there is a finite computation in which $\bar{b}$ does not occur after x. If the last node has a predecessor n' with the same label, then there is a cycle in the graph of the transducer in which no state q' occurs with $b\varepsilon\phi(q')$. Since every path on a repetition-free flowchart corresponds to a computation, there is an infinite computation with prefix x in which $\bar{b}$

does not occur after x. Hence not Ult$(b,\underline{x})$. This proves the claim and the theorem for the case of flowcharts.

The preceding theorem will now be used to show that there is at least some hope for obtaining the closure of a repetition-free, determinate, finite-state schema.

Theorem 13 Let $S=(A,T)$ be a repetition-free, determinate, finite-state schema. Let $\tilde{S}$ be the closure of S and $\tilde{\phi}$ the behavior of $\tilde{S}$. Then $\tilde{\phi}$ is total recursive in the following sense: For any string $x\epsilon\underline{A}^*$ it is decidable whether $\ulcorner\tilde{\phi}(x)\urcorner$, and if so then it is decidable for any $b\epsilon A$ whether $b\epsilon\tilde{\phi}(\underline{x})$.

Proof Referring to the proof of Theorem 11, note that the sequence of schemata $S=S_0,S_1,S_2,\ldots$ is such that each schema is finite-state and can be effectively constructed from its predecessor. Note also that this sequence adds prefixes which belong to the closure in order of increasing length. Therefore, for any given x, at some stage i either S_i will have x as a valid input or x will not be a valid input in the closure. In fact,

this stage i is such that $i \leq |\underline{A}|^{|\underline{x}|}$.

Remark Suppose that S is a finite-state schema and that somehow it is known that S possesses a finite-state closure. Consider the procedure of constructing the sequence of finite-state schemata $S=S_0,S_1,S_2,\ldots$ as in Theorem 11, stopping at each stage to test whether S_i is closed (using the local completeness criterion, for example). Hopefully this procedure would yield a closure of S_0 at some stage j, i.e. $S_j = \tilde{S}$. The example of Figure 8 shows that this is not the case. The flowchart S has the one-state closure as shown; however the procedure outlined produces the infinite sequence $S_1,S_2,\ldots$, none of which are closed. One case is known, however, when the procedure always converges. This is given in

Theorem 14 Let $S=(A,T)$ be a finite-state, determinate, repetition-free schema, where T is an acyclic transducer. A closure of S may be effectively constructed having these same properties.

Proof Since there is a uniform bound for the length of all computations of such a schema (and hence of any equivalent schema), the procedure of constructing $S=S_0,S_1,S_2,\ldots$ always reaches a stage where $S_i = S_{i+1}$ which is the closure of S.

Clearly there are finite-state schemata with cycles which are also closed; e.g. any schema for which $\rho=A\times A$ is closed. As motivation for considering transducers of greater complexity than finite-state, the following is presented.

Theorem 15 The repetition-free flowchart S_0 of Figure 9 has no finite-state closure.

Proof Consider the infinite family of finite-state schemata $\{S_n\}_{n\epsilon\omega}$ of Figure 9. A simple induction suffices to show $\forall n\epsilon\omega\ S_n \equiv S$. If $\tilde{S}$ is is a closure of S, then it must include the computations of every S_n. It is then easy to see that the domain of $\tilde{\phi}$, the behavior of $\tilde{S}$, includes the set $P=\{x\epsilon\{a_1,b_1\}^* \mid |E(\{a_1\},x)| \geq |E(\{b_1\},x)|\}$. For any $x\epsilon P$

$$\tilde{\phi}(x)=\begin{cases}\{a,b\} \text{ if } |E(\{a_1\},x)| > |E(\{b_1\},x)| \\ \{a\} \quad \text{if } |E(\{a_1\},x)| = |E(\{b_1\},x)|\end{cases}$$

If there is a finite-state transducer for $\tilde{S}$, then the equivalence relation $\approx$ on $\{a_1,b_1\}^*$ defined by $x \approx y$ iff $\forall w\epsilon\{a_1,b_1\}^*\ \tilde{\phi}(xw)=\tilde{\phi}(yw)$ must be of finite rank[13]. However this is not the case, since letting $x(m)=\underbrace{a_1\ldots.a_1}_{\text{m-times}}$, $n < m$ implies $x(n)\not\approx x(m)$, since $\tilde{\phi}(x(n)\underbrace{b_1\ldots.b_1}_{\text{n-times}})=\{a\}$; whereas $\tilde{\phi}(x(m)\underbrace{b_1\ldots.b_1}_{\text{n-times}})=\{a,b\}$. Hence $\approx$ is of infinite rank.

This is a possibly surprising result because it shows that although the number of operations is finite, an infinite amount of storage (in the transducer) may still be required to schieve maximal parallelism. It can be shown that there is a closure of S with a real-time counter transducer with one counter. However such schemata are not of sufficient generality to realize the closure of an arbitrary repetition-free finite-state schema, as the following shows.

Theorem 16 The repetition-free flowchart S of Figure 10 has no closure with a real-time counter transducer.

Proof From an argument similar to the previous example, it is easy to see that any $y\epsilon\{c_1,c_2,a_1,b_1\}^*$ such that $\forall x \leq y$ $|E(\{c_1\},x)| \geq |E(\{a_1\},x)|$ and $|E(\{c_2\},x)| \geq |E(\{b_1\},x)|$ is a valid input to the transducer of the closure. For any natural number n, Let X_n be the set of inputs

$$\{(c_1)^{r_1}\ c_2\ (c_1)^{r_2}\ c_2\ldots.(c_1)^{r_n}\ c_2 \mid$$

$r_1,r_2,\ldots.,r_n\epsilon\{1,2\}\}$, where $(c_1)^{r_i}$ means $\underbrace{c_1c_1\ldots c_1}_{r_i\text{-times}}$.

Now an appeal is made to the following definition and lemma from [17].

Definition For any natural number n, define an equivalence relation $\overset{n}{\approx}$ on $\underline{A}^*$ by $\forall x,y\epsilon\underline{A}^*\ x \overset{n}{\approx} y$ iff $\forall w\epsilon\underline{A}^*\ |w| \leq n$ implies $\tilde{\phi}(xw)=\tilde{\phi}(yw)$.

Lemma 17 (Fischer, Meyer, and Rosenberg[17]) If $\tilde{\phi}$ is realizable by a real-time counter transducer then there exist constants c,k such that for any $n\epsilon\omega$ the number of equivalence classes of $\overset{n}{\approx}$ does not exceed cn^k.

To continue the proof, let x,y be distinct words from X_n, i.e.

$$x=(c_1)^{r_1}\ c_2\ (c_1)^{r_2}\ c_2\ldots.(c_1)^{r_n}\ c_2 \text{ and}$$

$y=(c_1)^{r'_1} c_2 (c_1)^{r'_2} c_2 \ldots (c_1)^{r'_n} c_2$, all $r_i, r'_i \varepsilon \{1,2\}$. Since x and y are distinct, there is some i such that $r_i \neq r'_i$. Let i be the least such number. Assume without loss of generality that $r_i < r'_i$. Letting

$$z=(a_1)^{r_1} b_1 (a_1)^{r_2} b_1 \ldots (a_1)^{r_i},\ \tilde{\phi}(xz)=\{a,c\}$$

while $\tilde{\phi}(yz)=\{b,c\}$. Since each string in X_n is of length no greater than 3n, and noting from above that $|z| \leq |x|$ and $|z| \leq |y|$ for $x,y \varepsilon X_n$, it follows that $x \neq y$ implies not $x \overset{3n}{\approx} y$. It is obvious that $|X_n|=2^n$, so that the number of equivalence classes of $\overset{3n}{\approx}$ is at least 2^n. Now supposing that $\tilde{\phi}$ is realizable by a counter transducer, $\exists c,k$ such that $\forall n$ the number of classes is not greater than cn^k. Hence $2^n \leq c(3n)^k$. This implies there exist constants c',k such that $\forall n$ $2^n \leq c'n^k$. This is an obvious contradiction since $2^n/c'n^k$ is unbounded as n increases without limit.

A real-time realization of the closure of S can be obtained using a single "queue", i.e. first-in-first-out list which essentially records the outcomes of c. This is a special case of a real-time "queue transducer" which is basically a multi-tape Turing machine with one read head and one write head per tape. All heads move in the same direction. Transducers of this type are useful for realizing closures of a sizable class of repetition-free flowcharts[1]. Space does not permit a detailed description of this class. The essential property of this class which is exploited is that, in the closure, no two operations with more than one terminator will ever be simultaneously active. There is a procedure for deciding membership in this class[1]. Moreover the closure can be effectively constructed for members of this class.

The question of the sufficiency of real-time queue transducers for the realization of the closure of an arbitrary repetition-free flowchart is open as of this writing.

References

1. Keller, R.M., Closures of Parallel Program Schemata, University of California, Berkeley, 1970 (Unpublished thesis)

2. Karp and Miller, Parallel Program Schemata, Journal of Computer and Systems Sciences, Vol. 3, No. 2, May 1969

3. Gill, S., Parallel Programming, Computer Journal, Vol. 1, April 1958

4. Richards, P., Parallel Programming, Technical Operations Incorporated, Report No. TO-B 69-27, 1960

5. Conway, M. E., A Multiprocessor System Design, Proc. FJCC 1963

6. Dennis and Van Horn, Programming Semantics for Multiprogrammed Computations, CACM, Vol. 9, No. 3, March 1966

7. Keller, R.M., Analysis of Implementation Errors in Digital Computing Systems, Washington University Computer Systems Laboratory, Tech. Rept. No. 6, March 1968

8. Bernstein, A.J., Analysis of Programs for Parallel Processing, Trans. IEEE, Vol. EC-15

9. Yanov, Yu. I., The Logical Schemes of Algorithms, Problems of Cybernetics, Vol. 1, 1960

10. Luckham, Park, and Paterson, On Formalised Computer Programs, Journal of Computer and Systems Sciences, June 1970

11. Manna, Z. Propeties of Programs and the First-Order Predicate Calculus, JACM, Vol. 16, No. 2, April 1969

12. Kaplan, D.M., The Formal Theoretic Analysis of Strong Equivalence for Elemental Programs, Stanford University Tech. Rept. No. CS 101, June 1968

13. Harrison, M.A., Introduction to Switching and Automata Theory, McGraw-Hill, 1965

14. Dijkstra, E.W., Cooperating Sequential Processes, in Programming Languages, Genuys, ed., Academic Press 1968

15. Slutz, D.R., The Flowgraph Schemata Model of Parallel Computation, MIT Project MAC, Report MAC-TR-53 (Thesis), Sept. 1968

16. König, D., Theorie der Endlichen und Unendlichen Graphen, Akademische Verlagsgesellschaft, Leipzig, 1936

17. Fischer, Meyer, and Rosenberg, Counter Machines and Counter Languages, Mathematical Systems Theory, Vol. 2, No. 3

Acknowledgement

The author wishes to express his gratitude to Professor Richard M. Karp for numerous helpful suggestions concerning this material.

Table I Characterizations of Determinacy

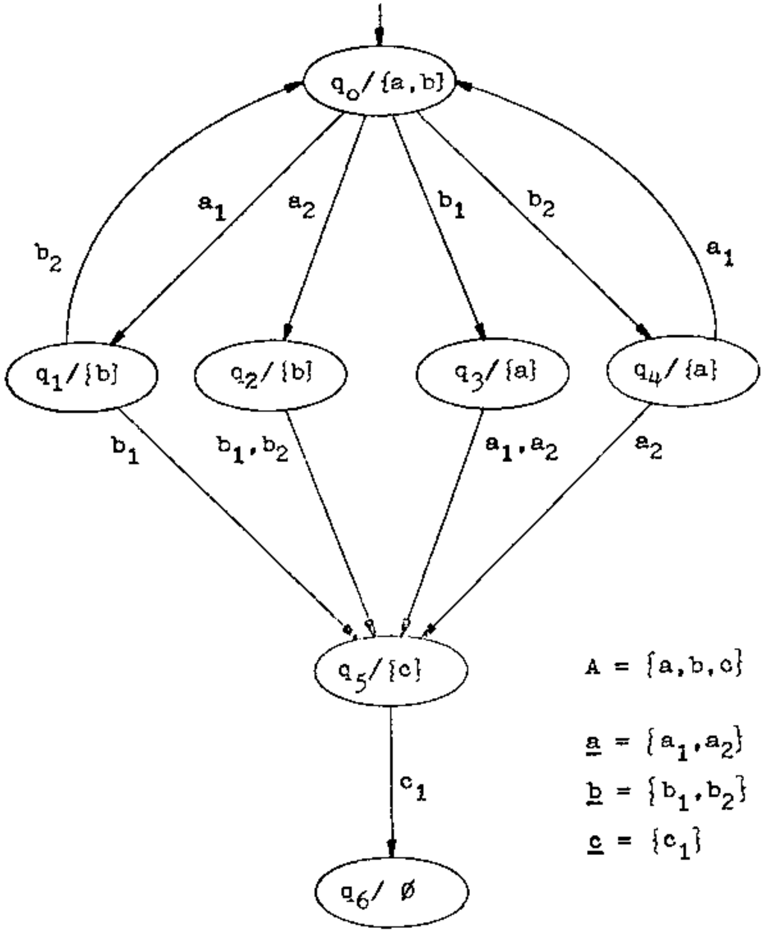
	Arbitrary schema	Commutative schema	Repetition-free schema
Conflict-free implies determinate	No	Yes	No
Determinate implies conflict-free	No	No	Yes
Syntactically determinate implies determinate	Yes	Yes	Yes
Determinate implies syntactically determinate	No	No	Yes

Table II Definition of $\hat{\phi}$, $\hat{f}$.

$$\forall_{q \varepsilon \hat{Q}} \quad \hat{\phi}(q) = \begin{cases} \phi(q) \cup \{b\} & \text{if } q \varepsilon Q^1 \\ \phi(q) & \text{if } q \varepsilon Q^2 \cup Q^3 \\ \phi(\bar{q}) & \text{if } q = \bar{q}^{\pi} \varepsilon Q^4 \end{cases}$$

$\forall_{q \varepsilon \hat{Q}} \quad \forall_{\gamma \varepsilon \underline{A}}$

$\hat{f}(q,\gamma)$	If $\gamma \varepsilon \underline{b}$	If $\gamma \notin \underline{b}$
If $q \varepsilon Q^1$	q^{γ}	$f(q,\gamma)$, if defined.
If $q \varepsilon Q^2 \cup Q^3$	$f(q,\gamma)$, if defined.	$f(q,\gamma)$, if defined.
If $q = \bar{q}^{\pi} \varepsilon Q^4$	undefined	$\begin{cases} f(\bar{q},\gamma)^{\pi} \text{ if } f(\bar{q},\gamma) \varepsilon Q^1 \\ f(\bar{q},\gamma\pi) \text{ if } f(\bar{q},\gamma) \varepsilon Q^2 \\ \text{undefined otherwise} \end{cases}$



$A = \{a,b,c\}$

$\underline{a} = \{a_1,a_2\}$

$\underline{b} = \{b_1,b_2\}$

$\underline{c} = \{c_1\}$

Verbal Description: Allow operations a and b to proceed simultaneously. If the paired outcomes of a and b are respectively (a_1,b_1), (a_2,b_1), or (a_2,b_2), then do c and stop. Otherwise start again from the beginning.

Fig. 1. Transducer T.

$\overset{a}{i \to j}$

means $D(a)=\{i\}$
$R(a)=\{j\}$

	D	R
a	{1}	{2}
b	{1}	{3}
c	{2,3}	{4}

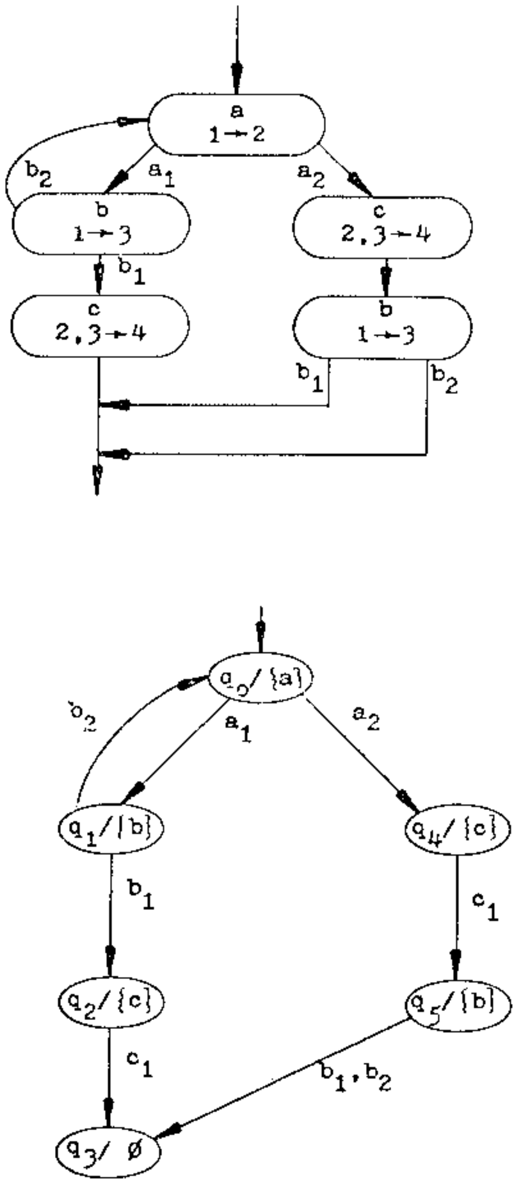
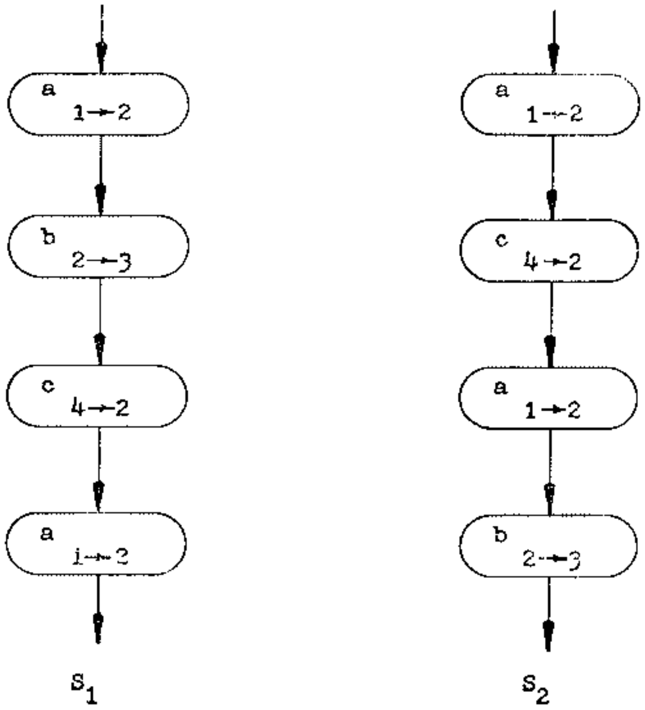


Fig. 2. The top diagram shows an alternate representation of the flowchart on the bottom.



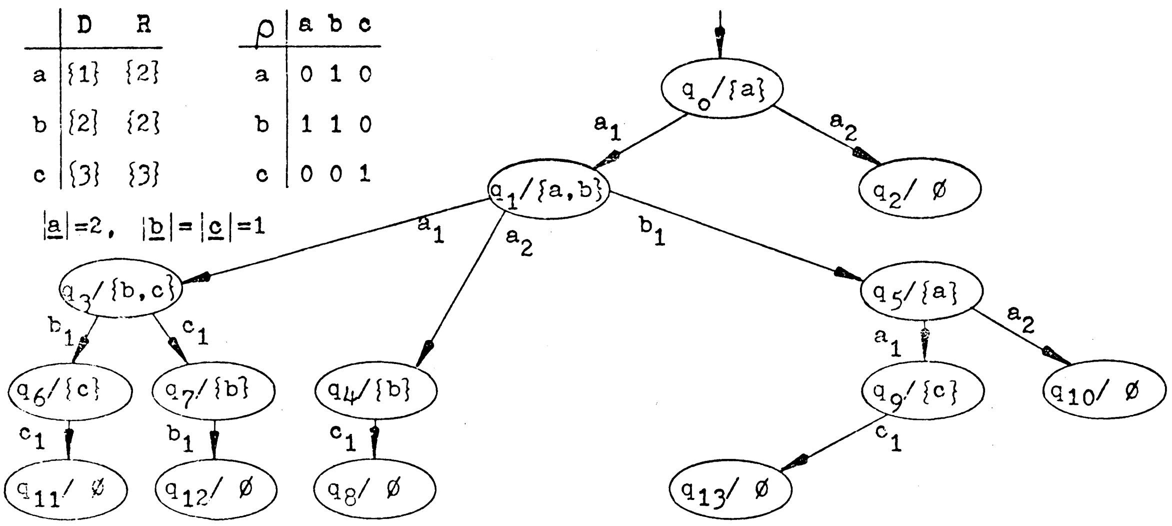
$\rho = (A \times A) - I$

Fig. 3. Non-repetition-free, equivalent flowcharts.

	D	R
a	{1}	{2}
b	{2}	{2}
c	{3}	{3}

ρ	a	b	c
a	0	1	0
b	1	1	0
c	0	0	1

$|\underline{a}|=2, \quad |\underline{b}|=|\underline{c}|=1$



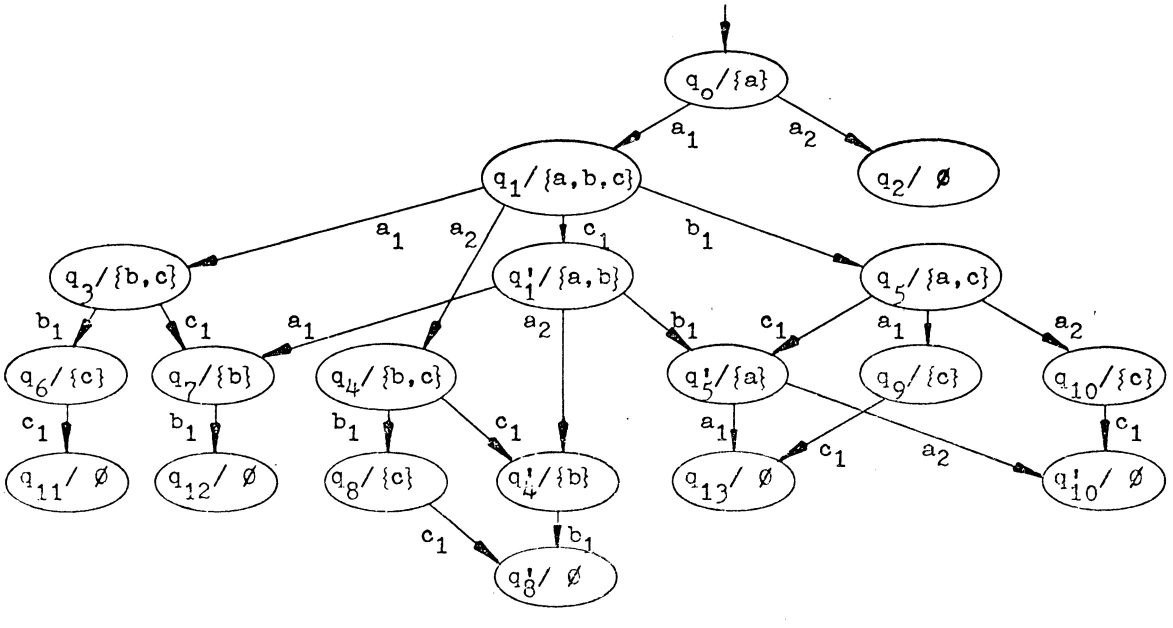
$\underline{\mathrm{Comp}(S)} = \{a_1a_1b_1c_1,\ a_1a_1c_1b_1,\ a_1b_1a_1c_1,\ a_2\} \Rightarrow \mathrm{Ult}(a_1,c).$

$Q^1 = \{q_1,q_4,q_5,q_8,q_{10}\}$

$Q^2 = \{q_3,q_6,q_9\}$

$Q^3 = \{q_o,q_2,q_7,q_{11},q_{12},q_{13}\}$

Fig. 4. A schema which is not globally complete.



$\underline{\mathrm{Comp}(\hat{S})} = \underline{\mathrm{Comp}(S)} \cup \{a_1b_1c_1a_1,\ a_1c_1a_1b_1,\ a_1c_1b_1a_1\}.$

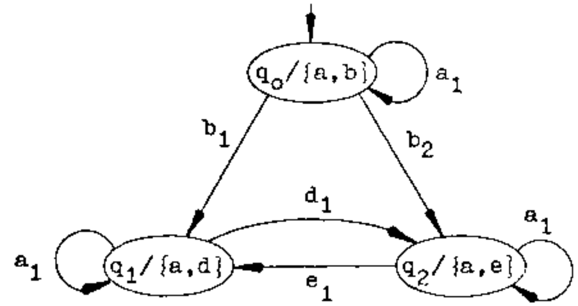
Fig. 5. Construction 1 applied to Fig. 4.

$A = \{a,b,d,e\}$

$|\underline{a}| = |\underline{d}| = |\underline{e}| = 1$

$|\underline{b}| = 2$

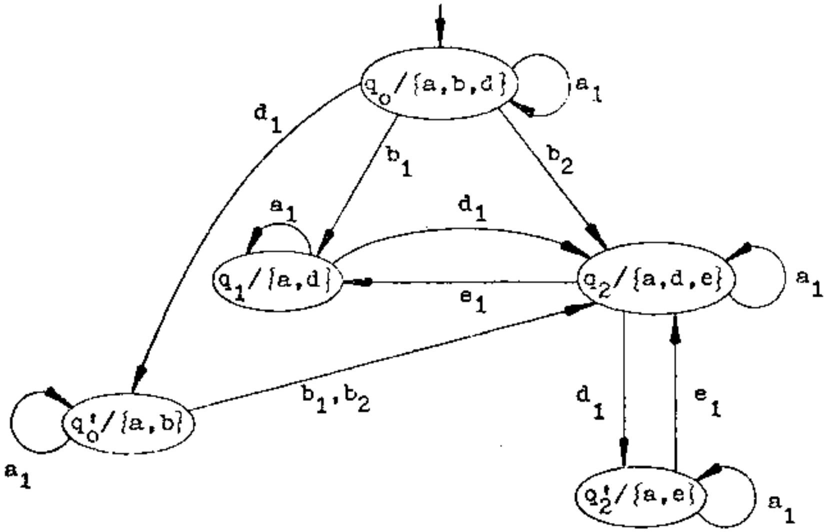
$\rho = I_A$



Schema S

Assume it is known that Ult(d,o).

Then $q_x = q_o$, $Q^1 = \{q_o, q_2\}$, $Q^2 = \{q_1\}$, $Q^3 = \emptyset$, $Q^4 = \{q_o', q_2'\}$.



Schema $\hat{S}$

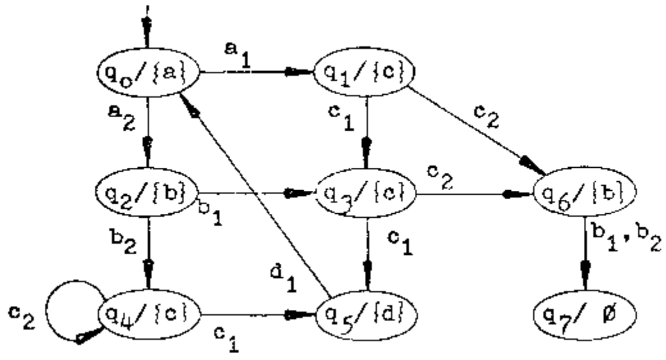
Fig. 6. Construction 1 applied to a commutative, finite-state schema.

$A = \{a,b,c,d\}$

$|\underline{a}| = |\underline{b}| = |\underline{c}| = 2$

$|\underline{d}| = 1$

$\rho = I_A$



Schema S

Let $x = \bar{a}\ a_1\bar{c}\ c_1\bar{c}\ c_1$ $q_x = q_5$.

Determine those operations e such that Ult(e,x).

Answer: $\{d,a,c\}$

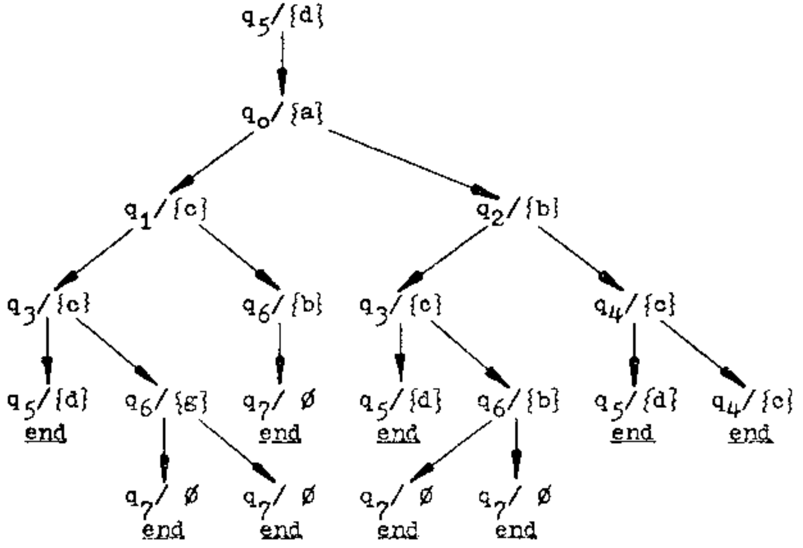
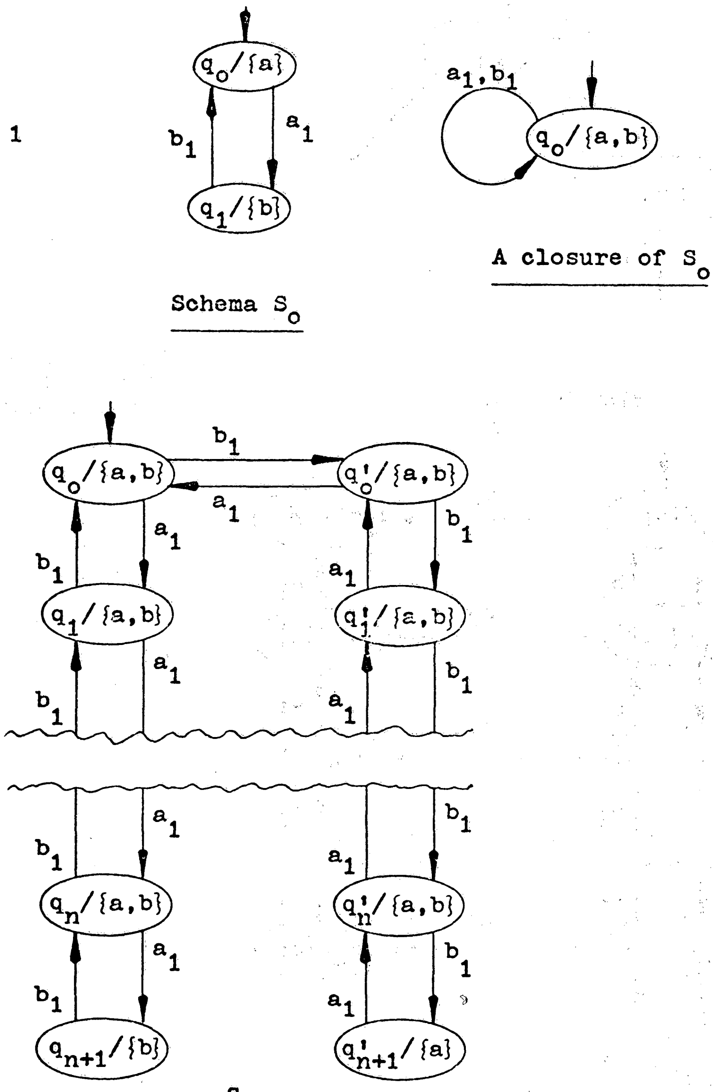


Fig. 7. Computing Ult for a repetition-free flowchart.

$A = \{a,b\}$

$|\underline{a}| = |\underline{b}| = 1$

$\rho = I_A$

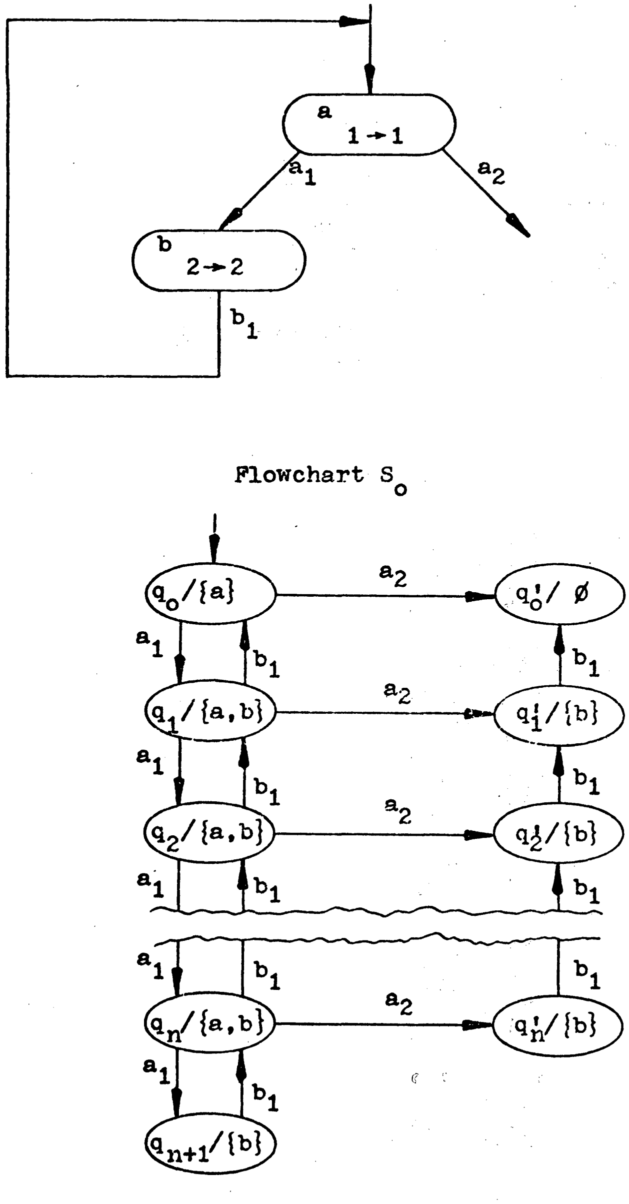


Schema S_o

A closure of S_o

S_{2n}

Fig. 8. The flowchart S_o has the single state closure shown. However, after 2n iterations of Construction 1, the result S_{2n} above is not closed.



Flowchart S_o

Schema S_n

Fig. 9. A flowchart with no finite-state closure. S_n is the n-th of infinitely many equivalents.

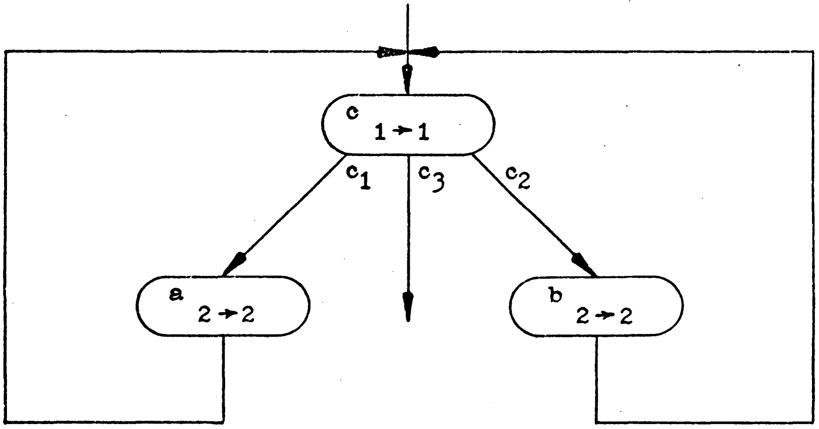


Fig. 10. A flowchart with no closure realizable by a real-time counter transducer.