MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project MAC

Machine Structures Group Memo No. 11 April, 1965

# Preliminary Description of Multi-Process Automata of Type A

Earl C. Van Horn

A multi-process automaton of type A (MPAA) is an abstract device consisting of a single processor, and a multi-process automaton structure of type A (MPASA, or structure). A structure is an ordered triple (P, M, W), where P is a set of operands, M is a set of microsteps, and W is a set whose content tells which microsteps are awaiting execution by the processor.

While each element of the set P is called an operand, the set P actually has the form of a mapping from a finite set of operand names into a set of operand properties.

$$P: n \rightarrow (\{0\} \times s) \cup (\{1, 2\} \times \omega \times s^{\omega})$$

Here n is the finite number of operand names, and s is the finite number of allowable operand states.

An operand is called a register, an inway, or an outway according to whether the first component of its property is 0, 1, or 2, respectively. Each register has two components in its property, the second of which is called a word. Inways and outways each have three components in their properties, the second being called a pointer, and the third being called a tape.

While each element of the set M is called a microstep, the set M actually has the form of a mapping from a finite set of microstep encodings into a set of microstep directors.

$$M : e \rightarrow (\{0\} \times n \times e^{*}) \cup (\{1\} \times n \times s \times e) \cup (\{2\} \times s \times e) \cup \{3\}$$

Here e is the finite number of microstep encodings. A microstep is called a get, put, fork, or terminate according to whether the first component of its director is 0, 1, 2, or 3 respectively.

The set W has the form of a mapping from the set of encodings into a set of counts.

$$W : e \rightarrow \omega$$

The count associated with an encoding gives the number of executions of the corresponding microstep which the processor must actually perform.

The processor of an MPAA operates by arbitrarily selecting an encoding having a non-zero count, subtracting one from its count, and executing the corresponding microstep. This execution will, in general, result in further changes in the structure of the MPAA. After completing the microstep execution, the processor selects another encoding with non-zero count, and carries out the same operation with respect to this new encoding. In this way, the processor steps the MPAA from one structure to another, by successive execution of individual microsteps.

In executing a get, the processor refers to the operand whose name is the second component of the get's director, and determines the state of this operand. If the operand is a register, its state is its word. If the operand is an inway, its state is the i-th component of its tape, where i is the value of its pointer.

After determining the state of an inway, the processor adds one to the inway's pointer. If the operand is an outway, its state is 0. Regardless of the type of the operand , the function which is the third component of the get's director is evaluated using as an argument the state that was determined from the operand. The value of this function is a new microstep encoding, to the count of which the processor then adds one.

Let us digress just briefly to define the function, $\tau$, which, when given an operand property, gives out a value which is the state of the corresponding operand, as determined by the algorithm given in the preceeding paragraph.

$$\tau : (\{0\} \times s) \cup (\{1, 2\} \times \omega \times s \overset{\omega}{\rightarrow} s$$

Note that $\tau$ is a function of the ordinary, passive sort; it does not add one to any pointer.

In executing a put, the processor refers to the operand whose name is the second component of the put's director. If the operand is a register, the processor replaces its word with the third component of the put's director. If the operand is an inway, it is not changed. If the operand is an outway, the processor replaces the i-th component of its tape with the third component of the director, and then adds one to the outway's pointer. Finally, regardless of the type of the operand, one is added to the count of the encoding which is the fourth component of the put's director.

In executing a fork, the processor does not refer to any operand, but adds one to the counts of the two encodings which are the second and third components of the fork's director.

In executing a terminate, the processor neither refers to an operand nor adds to any count.

Figure 1 depicts the nature of four basic types of microsteps.

The selection of an encoding with non-zero count is made in a completely arbitrary way. We assume that we do not even have any statistical knowledge about the selection operation.

An MPAA begins its operation with some initial structure. Its processor then executes microsteps until a structure is produced in which there are no non-zero counts, at which point the MPAA stops. Of course, it is possible for an MPAA to compute forever without stopping.

An initial structure can, of course, be any MPASA. However, in order to better understand the relationship between MPAA's and real-world computing systems, it is useful to examine a typical initial structure, and to define certain informal subdivisions of MPASA's in general.

In a typical initial structure, we can imagine that the set M constitutes the order code of the MPAA, and that this set together with the numbers n, s, and e constitute the machine design of the MPAA. The initial words make up a freshly loaded program. The set W represents the initial machine conditions of the MPAA, and a typical initial structure might have only one non-zero count. Let all pointers, as well as all components of every output tape, be initially equal to 0. The initial components of the input tapes are the input data of the MPAA, and if any input tape is to hold only a finite amount of data, we can let its unused components be equal to 0.

Notice that the words, the pointers, the components of the output tapes, and the counts are the only parts of the structure of an MPAA which are altered in the course of the MPAA's operation. The set M, the input tapes, and the types and number of the operands are never changed from their initial conditions.

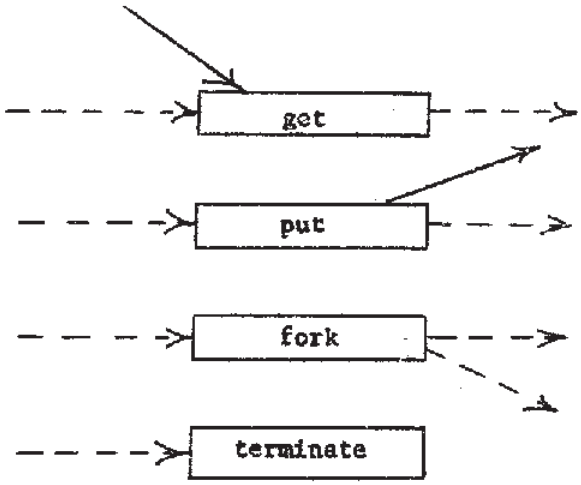


Figure 1. The four basic types of microsteps.
——→ : data flow - - - -→ : control flow

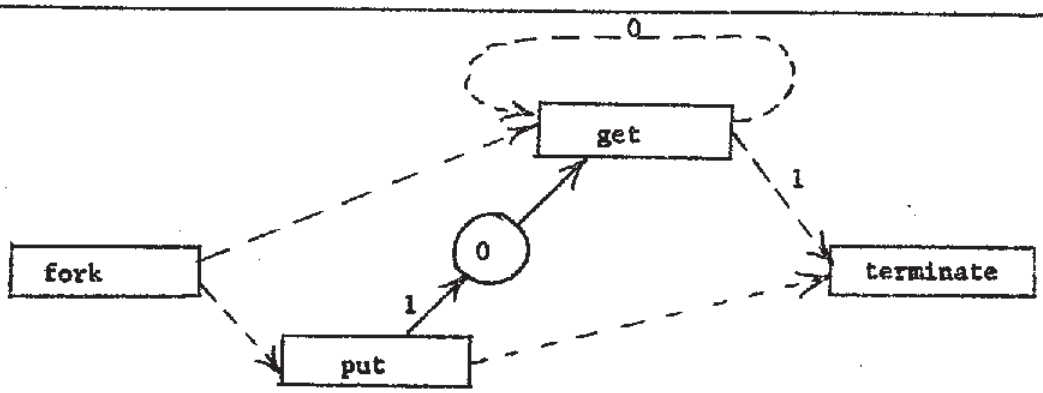


Figure 2. A sometimes-stopping MPASA.
The circle indicates a register. Its initial word is 0. n = 1, s = 2, e = 4.

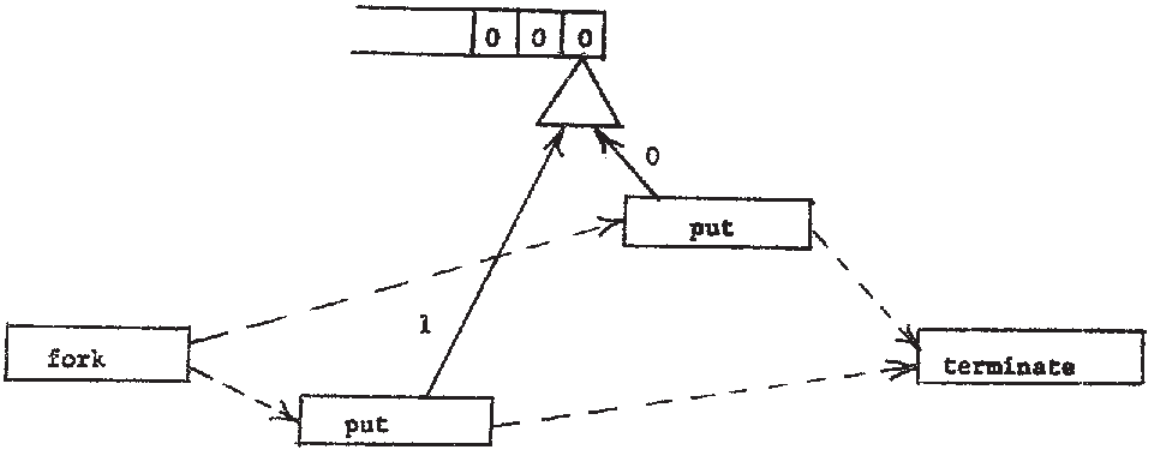


Figure 3. An ambiguous MPASA. The triangle denotes an output tape with the first three components initially equal to 0. n = 1, s = 2, e = 4.

Let us denote the set of all possible MPASA's by the symbol $\Sigma$.

$$\Sigma = ( (\{0\} \times s) \cup ( \{1, 2\} \times \omega \times s^{\omega}) )^{n} \times ( (\{0\} \times n \times e^{s}) \cup (\{1\} \times n \times s \times e) \cup (\{2\} \times e \times e) \cup \{3\} )^{e} \times \omega^{e}$$

We can now define the function

$$\pi : \Sigma \times e \longrightarrow \Sigma$$

which maps an input structure and an input encoding into the output structure which results from the processor's operation on the input structure after it has selected the input encoding. That is, $\pi$ models both the processor's action of subtracting one from the count of the input encoding and the processor's action of executing the microstep corresponding to the input encoding. If the count of the input encoding is equal to 0 in the input structure, no subtraction is performed and the output structure is equal to the input structure.

In order to model the encoding selection operation of the processor, we must postulate a random selection function

$$\text{ш} : \Sigma \longrightarrow e$$

which chooses arbitrarily an encoding which has a non-zero count in the input structure. If all of the encodings in the input structure habe zero counts, then the output encoding is equal to 0. The unusual letter, ш, is from the Russian alphabet and is pronounced "shuh". It is used here instead of a Greek letter in order to call attention to the fact that the random selection function is not really a function is the set-theoretic sense.

For a given initial structure, I, we can exhibit at least one <u>encoding sequence</u>, E, which is an element of $e^{\omega}$, and a corresponding <u>structure sequence</u>, S, which is an element of $\Sigma^{\omega}$, by means of the following equations.

$$S_0 = I \qquad (*)$$

$$E_i = /\!/\!/\ [S_i] \qquad (* *)$$

$$S_{i+1} = \pi\ [S_i, E_i] \qquad (* * *)$$

Because of the occurrance of /// in (* *), an initial structure can give rise to many encoding sequences, and hence many structure sequences. However, given an initial structure and an encoding sequence, we can determine a single structure sequence by making use of equations (*) and (* * *) only, in which equations /// does not appear. This relationship can be formalized by means of a function.

$$\sigma : \Sigma \times e^{\omega} \rightarrow \Sigma^{\omega}$$

Let us call an ordered pair of the form (I, E) a run. Then $\sigma$ maps a run into its corresponding structure sequence.

We consider the sequences E and S to be infinite even when the processor generating them stops after a finite number of microstep executions. In this case, the remaining components of E are zero, and the remaining components of S are equal to the final structure. Notice that such a pair of sequences may also be generated by an MPAA that does not stop. This might occur, for example, in an MPAA having a put whose own encoding is equal to 0 and whose director has a fourth component of 0.

Consider the MPAA whose structure is given below and depicted schematically in Figure 2.

$$n = 1$$

$$s = 2$$

$$e = 4$$

$$P = \{(0, (0, 0))\}$$

$$M = \left\{ \begin{array}{l} (0, (2, 1, 2)) \\ (1, (0, 0, \{(0, 1), (1, 3)\})) \\ (2, (1, 0, 1, 3)) \\ (3, (3)) \end{array} \right\}$$

$$W = \left\{ \begin{array}{l} (0, 1) \\ (0, 0) \\ (0, 0) \\ (0, 0) \end{array} \right\}$$

Notice that if the put is ever executed, the MPAA will stop, whereas if the put is never executed, then the MPAA will never stop. Since whether or not the put is executed depends on the caprice of the function $\mathcal{M}$, we cannot say whether or not an MPAA with this initial structure will stop.

In general, it is clear that some initial structures will always lead to a stop, that other initial structures will never lead to a stop, and that others will sometimes lead to a stop and sometimes not. We need to examine the encoding sequence that results from the actual operation of the MPAA, along with the initial structure, to determine whether or not the MPAA stops. It is then a trivial matter to determine whether or not a stop occurs. In other words, we must associate the property of stoping with a run, rather than with an initial structure.

Now consider the problem of determining whether or not an MPAA with a given initial structure will eventually write some specific number x in the i-th component of its j-th output tape. It is not hard to see that the $\mathcal{M}$ function will introduce the same sort of uncertainties into this problem as it did in the case of the stopping problem. For example, in the MPAA of Figure 3, the first

square of the output tape will receive a 0 or a 1 depending upon which put is executed first. It is clear that we must consider the output behavior of an MPAA as being a function of a run, not just an initial structure.

The examples given above motivate the following definitions.

Definition Two runs are i-j equivalent if and only if their initial structures are equal, and each puts the same number into the i-th component of its j-th output tape.

Definition Two runs are behaviorily equivalent if and only if they are i-j equivalent for all components of all of their output tapes.