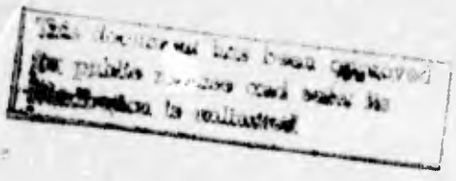
FINITE STATE QUEUE AUTOMATA

R. L. Bisbey

March 1969



P-4051

FINITE STATE QUEUE AUTOMATA

R. L. Bisbey†

The RAND Corporation, Santa Monica, California

This Paper examines the power of a finite-state machine equipped with a queue (FIFO list), henceforth called an FSQA (Finite State Queue Automata).

When examining mechanical language recognizers, one observes that the difference between recognizers of Context Free and Regular languages is the addition of a push-down stack (LIFO list) to a finite-state machine--such a machine will be called PDA, Push-Down Automata. The stack allows the finite-state machine to process languages that have phrases within other phrases, or what Chomsky [1] calls *self-embedding*. Examples of *self-embedding* include the Lukasiewicz (Polish) notation (Iverson [2]). The stack allows the finite-state machine to temporarily store symbols in it for later analysis, as in the case of the language xcx^* where x is a non-null string of a's and b's, and x^* is the mirror image of x. Recognition of languages using a stack involves a form of tree walking, which Knuth [3] calls *pre-order*; i.e.,

---

†Any views expressed in this Paper are those of the author. They should not be interpreted as reflecting the views of The RAND Corporation or the official opinion or policy of any of its governmental or private research sponsors. Papers are reproduced by The RAND Corporation as a courtesy to members of its staff.

This work is a result of a seminar organized by Barbara Partee at the University of California, Los Angeles.

1) Visit the root;

2) Transverse the sub-tree of the first tree;

3) Transverse the remaining trees.

Figure 1 illustrates such a tree walk.

Fig. 1—Tree walk using a push-down stack

An obvious question is: What language recognition capabilities can be achieved by adding a queue (FIFO list) to a finite-state machine? Several things may be noted immediately in this seemingly unexplored area:

1) A queue would allow the finite-state machine to analyze trees one layer at a time, as in Fig. 2.

2) The machine is more powerful than a finite-state machine and probably has many attributes of linear-bounded automata, since it can recognize languages of the form xcx where x is a string of a's and b's.

(Note that a PDA cannot even recognize this latter language.)

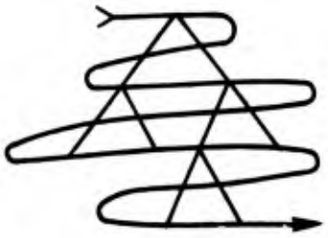


Fig. 2—Tree walk using a queue

The following is a characterization of a PDA used in Ginsburg [4]. An alternative definition may be found in Chomsky [5].

A PDA is a seven-tuple $M(k,\Sigma,\Gamma,\delta,Z_o,q_o,F)$ where

1) K is a nonempty finite set of states;
2) Σ is a nonempty finite set of inputs;
3) Γ is a finite nonempty set of push-down symbols;
4) δ is a mapping from $Kx(\Sigma U\{\varepsilon\})x\Gamma$ into the finite subsets of $kx\Gamma^*$;
5) Z_o is an element of Γ (the start push-down symbol);
6) q_o is in K (the start state);
7) F is a subset of K (the set of final states).

Informally, the machine is defined as an input symbol, a work-stack symbol, and a current state. The machine is allowed to read each input symbol only once, but it may elect not to read the input while in a given state. Processing on the work stack may be characterized by either of two operations: 1) a symbol may be written on the right (left) end of the tape; or 2) the right (left) symbol may be erased. A schematic diagram of a PDA is shown in Fig. 3.

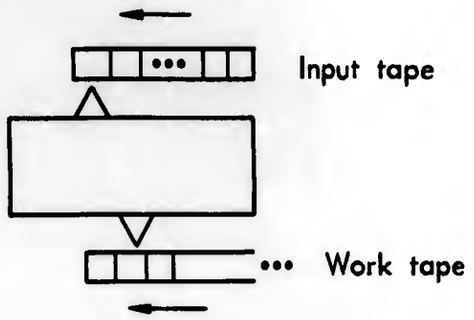


Fig. 3—Push-down automata

The definition of the FSQA will be identical to that of the PDA except for the operations that can be performed on the work queue. Processing on the work queue may be characterized by two operations: 1) a symbol may be written on the right (left) end of the tape; and 2) a symbol may be read from the left (right) end of the tape.

This definition indicates that once a symbol is written, it cannot be written over, and that once a symbol is read, it cannot be reread. Figure 4 shows a possible schematic diagram of an FSQA.

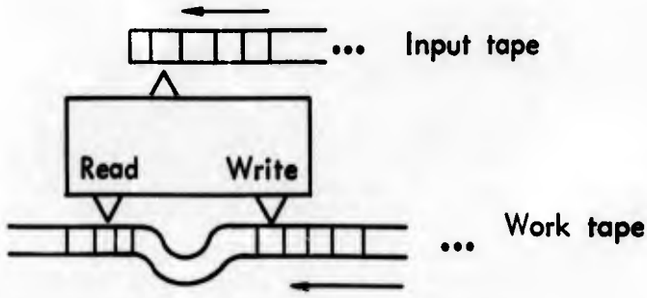


Fig. 4—Finite-state queue automata

The queue portion of the machine would have two tape heads, one for reading the work tape and one for writing the work tape. The tape would be of infinite length on both ends and would contain $n \geq 0$ symbols between the read and write heads. Various boundary conditions could be specified for either the PDA or the FSQA, but they are not essential to the following theorem and are thus left undefined.

The following theorem requires a new tape symbol λ in Γ to be used as a "marker."

THEOREM. An FSQA can perform all functions of a PDA.

Stack Addition: Adding a symbol to the top of the tape requires identical operations for the two machines.

Stack Deletion: For each rule $\{\phi^n{}_{,}{}_{,}\} \rightarrow \{\varepsilon,\phi^z\}$ that removes a symbol from the top of the tape of a PDA, the set of states $\phi^j_o,\ldots,\phi^j_i$ will be added to the FSQA, where i is the number of elements in Γ. The right side of the original deletion rule will be replaced by $\{\lambda,\phi^j_o\}$, while the remainder of the rules added will be defined by:

$\{\phi^j_o,\varepsilon,\chi\} \rightarrow \{\varepsilon,\phi^j_\chi\}$

$\{\phi^j_K,\varepsilon,\chi\} \rightarrow \{K,\phi^j_\chi\}$ $\quad K \neq 0,\ \chi \neq \lambda$

$\{\phi^j_K,\varepsilon,\lambda\} \rightarrow \{\varepsilon,\phi^z\}$ $\quad K \neq 0$, ϕ^z = next state in original unstacking rule

The right side of the last set of rules will be the same as the right side of the original unstacking rule in the PDA. The above rules will cause the following sequence to be executed in place of the original deletion rule:

1) Write λ on the top of the tape and go to state ϕ^j_o;
2) In ϕ^j_o reading x on the work tape, go to state ϕ^j_χ;
3) In ϕ^j_χ reading $y \neq \lambda$ on the work tape, go to state ϕ^j_y and write x on the tape;
4) In ϕ^j_χ reading λ on the work tape, go to the state indicated by the original deletion rule.

Stated more succinctly, for each deletion rule in the original machine, i states have been added (where i is the number of possible stack symbols) and have been arranged to "remember" the previous symbol read from the work tape.

LEMMA. An FSQA can perform all the functions of a PDA that has two stacks.

The lemma is proved by applying the above theorem twice.

As a result of the above theorem and lemma, the conclusion can be reached that an FSQA has the same power as a Turing machine by noting that a PDA with two stacks is equivalent to a Turing machine--the proof originally appearing in McCarthy [6] and later in Fischer [7]. This makes an FSQA uninteresting from a linguistic viewpoint since Turing machines are equivalent to Unrestricted Rewriting Systems.

REFERENCES

1) Chomsky, N., and G. Miller, "Introduction to the Formal Analysis of Natural Languages," *Handbook of Mathematical Psychology*, Vol. 2, Luce, Bush, and Galanter (eds.), Wiley, New York, 1963, pp. 269-322.

2) Iverson, K., *A Programming Language*, Wiley, New York, 1962.

3) Knuth, D., *The Art of Computer Programming*, Addison-Wesley, New York, 1968.

4) Ginsburg, S., *The Mathematical Theory of Context Free Languages*, McGraw-Hill, New York, 1966.

5) Chomsky, N., "Formal Properties of Grammars," *Handbook of Mathematical Psychology*, Vol. 2, Luce, Bush, and Galanter (eds.), Wiley, New York, 1963, pp. 323-418.

6) McCarthy, J., *Recursive Functions of Symbolic Expressions and their Computation by Machine*, Quarterly Rpt. #53, Laboratory of Electronics, M.I.T., 1959.

7) Fischer, P., "Turing Machines and Restricted Memory Access," *Information and Control*, Vol. 9, 1966, pp. 364-379.