CIRCULAR AUTOMATA

Charles Zaiontz, University of South Florida

ABSTRACT

We define a finite-state machine called a circular automata (CA) which processes information in a queue; we show that any function computed (or any language recognized) by such a machine is computable (recognizable) by a Turing machine and vice versa. Space and time bounds are given for the needed simulations. Furthermore, the class of languages recognized by (non-) deterministic linear bounded automata is equal to the class of languages recognized by (non-) deterministic CA which don't expand the length of the contents of the queue. Whether every language recognized by such a non-expanding CA is recognized by a deterministic one is equivalent to the famous LBA problem.

CA can be viewed as generalizations of ordinary finite automata and as a Shepherdson-Sturgis single register machine programming language. An interesting model of a non-expanding CA is that of a finite-state machine which process tapes in the form of a loop. This appears to be a very natural way to process magnetic tape which circles back on itself.

INTRODUCTION

A circular automata (CA) is a machine with finite control which operates on a queue (or tape or register) which is potentially infinite on the right. In processing information the leftmost symbol in the queue is read. Depending upon this symbol and the current internal state of the machine, additional symbols may then be added onto the right end of the queue. The leftmost symbol then leaves the queue. Thus each move of the CA consists of reading and ejecting a symbol on the left and entering new information on the right. A non-expanding circular automata is a CA in which exactly one new symbol is written in any given move.

Formally, a non-deterministic CA (NDCA) is a tuple $M = (K,\Sigma,\Gamma,q_0,q_f,\delta)$ where K,Σ,Γ are finite alphabets, K is the set of (internal) states, Γ is the set of tape symbols, $\Sigma \subseteq \Gamma$ is the set of input symbols, $q_0 \in K$ is the initial state, $q_f \in K$ is the final state, and $\delta : K\times\Gamma \to \mathcal{P}(K\times\Gamma^*)$ is the transition function. A circular automata is deterministic (written DCA) if for each $q \in K$, $a \in \Gamma$, $\delta(q,a)$ is a singleton. In this case we may consider $\delta : K\times\Gamma \to K\times\Gamma^*$. Note that if M is a deterministic non-expanding CA then we may consider $\delta : K\times\Gamma \to K\times\Gamma$.

An instantaneous description (ID) of M is a sequence qw where $q \in K$ and $w \in \Gamma^*$. ID I directly produces ID J, denoted $I \vdash_M J$, if I is of the form qau and J is of the form $q'uv$ where $q,q' \in K$; $a \in \Gamma$; $u,v \in \Gamma^*$; and $(q',v) \in \delta(q,a)$. I eventually produces J, $I \vdash_M^* J$, if there are $I_0, I_1, \ldots, I_n$ with $I = I_0 \vdash_M I_1 \vdash_M \ldots I_n = J$. An input $w \in \Sigma^*$ is accepted by M if $q_0 w \vdash_M^* q_f u$ for some $u \in \Gamma^*$. The language recognized by M is the set of all input words accepted by M. Although we won't do it here, CA may also be used to compute functions.

A CA may be regarded as a machine which processes a one-way infinite tape in the manner of an ordinary finite automata, but with the added ability of writing symbols on the right end of the non-blank portion of its tape. Using this model of a CA, an ordinary finite automata is just a non-writing CA. A CA can also be viewed as a Shepherdson-Sturgis single register machine with

instructions JMP a,N and DAA w where N is an instruction label, $a \in \Gamma$, and $w \in \Gamma^*$. Here JMP a,N means jump to instruction N if the leftmost symbol of the register is an "a", and DAA w means delete the leftmost symbol in the register and add w onto the right end of the (non-blank portion of the) register.

CIRCULAR AUTOMATA AND TURING MACHINES

In this section we show how to simulate the actions of a Turing machine (TM) by a CA and vice versa. A (one-tape, one-way) non-deterministic TM (NDTM) may be defined as a tuple M exactly as for CA except that $\delta : K \times \Gamma \cup \{B\} \to \mathcal{P}(K \times \Gamma \times \{L,R\})$ where $B \notin \Gamma$ is the blank and L and R stand for move left and move right. ID's are of form uqv where $u,v \in \Gamma^*$ and $q \in K$. $I \vdash_M J$ if ID I becomes ID J after one move of M. $I \vdash_M^* J$ is defined as for CA. The language recognized by M is the set of all those input $w \in \Sigma^*$ which are accepted by M, i.e. for which $q_0 w \vdash_M^* u q_f v$ for some $u,v \in \Gamma^*$. The notions of deterministic TM (DTM) and function computable by a TM are as usual.

The simulation of a CA by a TM is easy. For example, the operation $qau \vdash_M q'uv$ of a CA M can be accomplished by a TM T with au on its tape as follows. T first erases the leftmost symbol ("a" in this case) and then moves its read head right until the last non-blank square is hit at which point v is added on at the end. The read head then moves left back to the leftmost non-blank square. In this way we get

Theorem 1: Any function computed (or any language recognized) by a (N)DCA can be computed (recognized) by a (N)DTM. Furthermore if the (N)DCA has time complexity $T(n) \geq n$ and space complexity $S(n)$, then the (N)DTM has time complexity $O(T^2(n))$ and space complexity $S(n)$.

The converse of this theorem is a little more difficult and requires the simulation of left and right circular shift operations by CA. Note that if $(q',a) \in \delta(q,a)$ for all $a \in \Gamma$ in a CA M, then whenever M gets into state q it performs a left circular shift on the current contents of its queue and then enters state q'. It is harder for a CA to perform a right circular shift. We now give an algorithm which shows how to compute $qwa \vdash_M^* q'aw$ by a non-expanding DCA M for any $a \in \Gamma$ and any $w \in \Gamma^*$ of length ≥ 1. If M is in state q then:

A. Place a marker "-" over the leftmost symbol in the queue and then (circular) shift left.

B. Place marker "-" over the scanned symbol and shift left twice.

C. (Test) If "-" is over the scanned symbol erase "-", keep shifting left until "-" is reached, erase "-", shift left, go into state q', and RETURN.

Otherwise, keep shifting left until "-" is reached, erase "-", shift left, GO TO B.

For example, suppose we wish to perform a right circular shift on input abcd using the above algorithm. The following are the steps in the computation, noting which instruction is being used:

A abcd B bcdā cdāb C dābc ābcd bcdā

B cdāb dābc C ābcd bcda cdab dabc

For input of length ≥ 3 the test instruction C is hit n-2 times and so $(n+1)(n-2)+1 = n^2-n-1$ steps are needed to perform the right circular shift using the above algorithm. Note that for each $a \in \Gamma$ we need a new tape symbol $\bar{a}$, a doubling of the total number of tape symbols. If desired we can get by with just one extra tape symbol "-", but at the cost of many etra states. Instead of placing a "-" above a symbol in Γ in the above algorithm just replace the symbol by the "-". Extra states are needed to keep track of the symbols replaced in this way. Another approach is simply to use the two symbols "a-" instead of $\bar{a}$, but this has the disadvantage of increasing the length of the non-blank portion of the queue. Using the above hints we get the following lemma

Lemma 1: For every set of tape symbols Γ one can construct a non-expanding DCA M which can perform a right circular shift of one square on any $w \in \Gamma^*$ in $O(n^2)$ time. We may further require that the set of tape symbols of M contain just one extra symbol. One can also construct a non-expanding DCA M which can compute $qawb \vdash_M^* q'bcw$.

Given $a,b,c \in \Gamma$ and $w \in \Gamma^*$, an M as defined by the last line of the lemma may be constructed as follows. If M is in state q and "a" is the leftmost symbol in the queue, then M follows the above algorithm with step A replaced by

A'. Delete "a" on the left and add "$\bar{c}$" on the right.

We now show how to define a CA M which can simulate the actions of a given TM T. If T is to process an input $w \in \Sigma^*$ then w is put in M's queue. Before M does anything else it puts a marker "-" over the last symbol in w. (For the present we may regard this marker as different from the one used in proving lemma 1.) This may be accomplished by having M perform a right circular shift on w, obtaining aw' where w is w'a. Then M puts a "-" over the leftmost symbol in its queue and performs a shift left, i.e. it deletes "a" on the left and adds $\bar{a}$ on the right, obtaining w'$\bar{a}$.

If at any point in its computations T has ID uqva then M will have ID qv$\bar{a}$u. Thus the leftmost symbol in M's queue tells where T's read head is located and the marker "-" tells where the right end of the non-blank portion of M's tape is. If T has ID uq, i.e. if T has moved right off the non-blank portion of its tape, then M will have ID q$\bar{B}$u. For convenience we use $\bar{u}$ to denote v$\bar{a}$ if u is a string of form va. We now show how to define M so that if $I \vdash_T J$ then $I' \vdash_M^* J'$ where I',J' are the simulations of the ID's I,J as defined above. This is accomplished using six cases. Let Γ be the set of tape symbols of T; let $a,b,c \in \Gamma$; $u,v \in \Gamma^*$ with v not the empty string.

1. If $uqav \vdash_T ubq'v$ then define M so that $qa\bar{v}u \vdash_M q'\bar{v}ub$, i.e. define the transition function of M so that $(q',b) \in \delta(q,a)$.

2. If $uqa \vdash_T ubq'$ then define M so that $q\bar{a}u \vdash_M^* q'\bar{B}ub$ as follows. If M is in state q and the leftmost symbol in its queue is $\bar{a}$ then delete the $\bar{a}$ on the left and add $b\bar{B}$ on the right. M now circular shifts right and goes into state q'. Alternatively, M deletes $\bar{a}$ on the left and adds $\bar{b}\bar{B}$ on the right; M then keeps shifting left until the $\bar{b}$ is scanned; M then deletes the $\bar{b}$ on the left and adds b on the right and goes into state q'

3. If $uq \vdash_T ubq'$ then we may define M so that $q\bar{B}u \vdash_M^* q'\bar{B}ub$ in a manner similar to case 2 above.

4. If $ucqav \vdash_T uq'cbv$ then define M so that $qa\bar{v}uc \vdash_M^* q'cb\bar{v}u$ using the last part of lemma 1.

5. If $ucqa \vdash_T uq'cb$ then define M so that $q\bar{a}uc \vdash_M^* q'c\bar{b}u$ using lemma 1 again.

6. Suppose T has ID qu, i.e. T is in state q scanning the leftmost symbol on its tape the non-blank portion of which is u. The corresponding ID of M is then $q\bar{u}$. Thus the rightmost symbol in M's queue has the "-" over it. This is the only situation in which this can occur. If T is in this situation and is called upon to shift left off the tape then processing by T will halt and the input will be rejected. Thus whenever T is called upon to move left (as in cases 4 and 5), M must first check to see if T is shifting off its tape. It does this by shifting the contents of its queue right; if the marker "-" is over the scanned symbol then the input is rejected (by cycling in some dead-end state for example), but if not then the queue is shifted left again back to its original position.

The above simulation by M requires many additional tape symbols besides those in Γ. But just as was the case in lemma 1 it is possible to alter the above so that only one extra tape symbol is needed. We have thus proved the following theorem.

Theorem 2: Any function computed (or any language recognized) by a TM T can be computed (recognized) by a CA. This holds for both deterministic and non-deterministic machines. In fact we may suppose that the transition function δ of M is such that if $(q',u) \in \delta(q,a)$ then u has length ≤ 2. Furthermore if T has time complexity $T(n) \geq n$ and space complexity $S(n)$ then M has time complexity $O(T^3(n))$ and space complexity $S(n)$.

Thus the class of languages recognized by CA is just the class of all r.e. languages and the class of functions computable by CA is just the class of recursive functions. Since computations by NDTM can be simulated by DTM we get

Corollary 1: Any function computed (or any language recognized) by a NDCA can be computed (or recognized) by a DCA.

NON-EXPANDING CIRCULAR AUTOMATA AND LINEAR BOUNDED AUTOMATA

Note that in the case where the TM is a linear bounded automata (LBA), the CA used in the above simulation is restricted to cases 1,4,and 5 (a case like #6 is actually needed on both the right and left extremities of the tape) and so it never increases the length of the contents of the queue. We therefore get the following corollary to theorems 1 and 2.

Corollary 2: The class of languages recognized by a (N)DLBA is equal to the class of languages recognized by a non-expanding (N)DCA.

If we take the queue of a CA and bend it into an annulus we get the idea for a particularly interesting model of a non-expanding CA. In this way a non-expanding CA becomes a finite-state machine with a fixed read head which processes input tapes which are in the form of a loop. In processing such a loop a symbol is scanned, depending on this symbol and the current state of the machine the scanned symbol may be changed and the machine goes into a new state. The tape is then shifted one square counter-clockwise. A loop with designated start square is accepted if the processing begins in that

square with the finite control in the initial state and eventually the finite control enters the final state. Of course it is easy to see that if an input of length n is accepted by a non-expanding CA with k states and m tape symbols, then it is accepted in $\leq km^n$ amount of time. Clearly we get an equivalent formulation if the processing always proceeded clockwise.

The above model is a very reasonable way of computing on a loop. By Corollary 2 the above is also a model of an LBA. The famous unsolved problem about LBA now takes the following form

Problem 1: Is the class of languages recognized by a non-expanding NDCA (or non-deterministic machine as given above) equal to the class of languages recognized by non-expanding DCA (or deterministic machine as in the above)?

Using the above corollary we also have the following characterization of LBA in terms of Shepherdson-Sturgis single register machines.

Corollary 3: The class of languages recognized by a non-expanding NDCA (or NDLBA ,i.e. the context-free languages) is equal to the class of languages accepted by a Shepherdson-Sturgis single register machine with instructions JMP a,N and DAAa where a is a single tape symbol.

REFERENCES

1. J.E. Hopcroft and J.D. Ullman, Formal Languages and Their Relation to Automata, Addison-Wesley, Reading Mass, 1969.

2. A.V. Aho, J.E. Hopcroft, and J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass, 1974.

3. J.C. Shepherdson and H.E. Sturgis, Computability of Recursive Functions, J.ACM 10:2, 1963, pp. 217-255.