MOORE REDUCIBILITY FOR REGULAR LANGUAGES

Paul Cull

Received September 25, 2008

Abstract. We investigate a simple notion of reduction for Regular sets. We call this *Moore* reduction because the transformations use Moore machines. We show that under this reduction the Regular sets form a bi-hierarchy with complete sets.

1 Introduction When is one regular language simpler than another? There are a variety of answers based on various metrics for various representations of the languages. For example: [2] [8] [11]

(a) number of states in the minimal automaton,
(b) star height of the regular expression,
(c) number of loops in the state diagram,
(d) number of loops in the circuit diagram,
(e) number of neurons in the neural net diagram,
(f) loop depth in the circuit or neural net diagram,
(g) etc.

In each of these representations, there is reasonable idea of "minimal" and a language cannot have a representation with a smaller value than its minimum value for the metric. Each of these metrics gives a hierarchy for regular sets. Sets at a higher level cannot be represented with only the resources for a lower level. It seems obvious that there cannot be any complete regular sets because each of these metrics gives values in the natural numbers, and so there cannot be an upper bound on the complexity of all regular languages.

Most complexity classes other than the regular languages are usually described with some notion of "reducibility". For example, the BIG open question $?\mathcal{P} = \mathcal{NP}?$ is usually described using polynomial time reducibility. [5] We will investigate some notions of reducibility for regular languages.

In a recent course, I made an off-hand remark that if a class of sets had a corresponding function class, then any non-trivial set in the class is complete for the class when the notion of reduction is many-one reduction with transformers from the function class. When I attempted to demonstrate this claim for regular languages, I ran into some problems and discovered a reasonable notion of regular reduction under which the regular sets form an infinite hierarchy but also have complete sets.

1.1 Trivial Reductions Question about reductions for regular languages is usually regarded as trivial because all nontrivial regular languages are equivalent with respect to certain notions of reduction. A language is nontrivial if it is neither the null set nor the universal set.

For example, the notion of *regular* Turing reduction

$$A \leq^{\mathrm{T}}_{\mathrm{reg}} B$$

would mean that a finite state recognizer for B could be used to build a finite state recognizer for A. So, if A and B are regular languages, then this relation holds in both directions. Nontriviality of B is not needed for Turing reductions.

A similar result holds for a version of *many-one* reduction which I'll call *constant space Turing machine many-one* reduction. In this reduction the reducing machine is a Turing machine with an input tape, a fixed size work tape (or equivalently *no* work tape), and an output tape. For any regular language A and any nontrivial regular language B, there is such a Turing machine which reduces A to B. This operation is very simple. The AB Turing machine simply simulates M_A (a finite state recognizer for A) on an input string x. This can be done in fixed finite (or no) work space since the simulating machine only needs to remember one of the finite number of states of M_A. At the end of x, the AB machine knows by the state of M_A it is remembering whether or not x is accepted by M_A. If x is accepted, then the AB machine outputs a string in B, say $b_{\rm yes}$, and if x is not accepted, then the AB machine outputs a string not in B, say $b_{\rm no}$. Obviously, if we let $T(x)$ be the output of AB on input x, then

$$x \in A \quad \text{iff} \quad T(x) \in B$$

and we have an easy to compute many-one reduction from A to B. (Note that the AB machine only reads the input once, while some notions of transduction would allow multiple reads of the input.)

While this may seem reasonable, we claim that it is really based on a "trick" which violates our idea of finite state reduction. The "trick" is that the AB machine can wait until and sense the **end** of the input. AB does not produce an output until it has found the end of the input. We think that this is unreasonable. In a more reasonable interpretation of finite state reduction, we would expect the transforming machine to actually work sequentially by outputting a string based on what it has seen so far and not based on the rest of the input which it has not yet seen.

2 Finite State Reductions The usual simplest complexity class is the **Regular** languages which are the set of strings which are recognized by **finite automata**. We want to consider finite state reductions of such languages.

2.1 Finite State Mappings A machine takes as input a string over some alphabet and outputs a string over the same or a different alphabet. We want to discuss the string transformations which are possible when the machine has a finite memory. (We are modeling a real computer with its finite internal memory. Unbounded external memory is not included in this model.) A machine with this memory limitation is usually called a **finite state machine**. Such a machine is described by specifying the input and output alphabet, the finite number of states, and two functions – the next state function and the output function.

The **next state function** calculates the new state given the present state and the present input symbol. This can be symbolized as

$$q_{t+1} = F(q_t, s_t)$$

where q_{t+1} is the state at time $t+1$ and it is determined from q_t, the state at time t, and s_t, the input symbol at time t. We call $F(q,s)$ the *next state function.*

The output of the machine could depend on the present state or on the present state and the input symbol. It turns out that either of these two choices for the output function will lead to essentially the same mappings and so we will choose the slightly simpler convention that the output is only a function of the present state. [3] A machine with this output convention is called a *Moore machine.* [9] Such machines are usually allowed to have only one output symbol for each time step, but we will find it more convenient to allow an output string for each time step. Symbolically, we write

$$w_t = G(q_t)$$

and mean that if the machine is in state q_t at time t, then it outputs the string w_t.

A *Mealy* machine has its output depending on both the state and the current input symbol. [3] In particular, this means that a Mealy machine gives no output for the null string. For a Mealy machine with state set Q and output alphabet Ω, we can define a Moore machine with state set $Q \times \Omega^*$, so that this Moore machine transitions to state (q, w) when the Mealy machine transitions to state q while outputting string w. Clearly, this Moore machine will produce the same output as the Mealy machine for any nonnull input sequence. For the null sequence, we give the Moore machine the initial state (q_0, Λ) so it will produce a null output just like the Mealy machine. As defined, the Moore machine has an infinite number of states, but the Moore machine only uses those states reachable from its initial state and this submachine is a finite state machine because the number of its states is bounded by the product of the number of states and input symbols for the Mealy machine.

2.2 Homomorphism This may be a reasonable place to mention homomorphism. A **homomorphism** is a mapping h,

$$h : \Sigma \to \Omega^*$$

which preserves concatenation. That is,

$$h(xy) = h(x \cdot y) = h(x) \cdot h(y) \text{ for all } x \text{ and } y,$$

and in particular, $h(\Lambda) = \Lambda$. Such a mapping corresponds to a Mealy transformation computable by a one state Mealy machine.

It is known that the Regular sets are closed under both homomorphism and inverse homomorphism. [3] Further, CFL (the class of context free languages without the null string) has a complete set with respect to homomorphic many-one reduction. [10]

On the other hand, there can be no complete regular sets with respect to homomorphism because sets requiring more states cannot be homomorphically reduced to sets requiring fewer states. Specifically, if the minimal machine for A has more states than the minimal machine for B, then any homomorphism will map non-equivalent strings, say x and y, for A to equivalent strings $h(x)$ and $h(y)$ for B, i.e. $F_A(q_0, x) \neq F_A(q_0, y)$ and $F_B(q_0, h(x)) = F_B(q_0, h(y))$. But, then by the homomorphism property, $h(xw) = h(x) \cdot h(w)$ and $h(yw) = h(y) \cdot h(w)$, and so

$$\begin{aligned} F_B(q_0, h(xw)) &= F_B(F_B(q_0, h(x)), h(w)) = \\ & F_B(F_B(q_0, h(y)), h(w)) = F_B(q_0, h(yw)). \end{aligned}$$

Hence, choosing w so that $xw \in A$ and $yw \notin A$ will give either both $h(xw)$ and $h(yw)$ are in B, or both $h(xw)$ and $h(yw)$ are not in B, violating the conditions for a valid reduction.

2.3 Moore Transformations At the moment, both F and G deal with one symbol at a time. We want to extend these functions so that they become functions of the whole input string. To do so, we define F^* which is a mapping from input strings to sequences of states:

$$F^*(q, x) = \begin{cases} q & \text{if} \quad x = \Lambda \text{ (Null string)} \\ q\, F^*(F(q, s), y) & \text{if} \quad x = sy \end{cases}$$

where s is a single input symbol and x and y are input strings. In fact, F^* gives us several mappings, one for each state q. If we specify one state, say q_0, as the initial state, then

$F^*(q_0, x)$ is the state sequence mapping for the machine, and it maps an input string $x = x_1 \ldots x_n$ to a sequence $\langle q_0, q_1, \ldots, q_n \rangle$ of states. For convenience, we'll also use $F(q, x)$ to mean the last state in the sequence $F^*(q, x)$.

Since the output function maps each state to an output sequence, the overall output produced by the input string x is

$$G(q_0) \cdot G(q_1) \cdot \ldots \cdot G(q_n)$$

where $G(q_i)$ is the output string corresponding to the state q_i and $\cdot$ indicates that all of these strings are being concatenated together. We can represent this entire concatenated string symbolically as

$$G^*(x)$$

and think of G^* as mapping input strings to output strings. Notice that G^* does not map strings of length n to strings of length n. Because $G(q)$ is a string and may possibly be the null string, the concatenation of $n+1$ such strings may produce a string shorter or longer than n characters. Of course, if we knew $m = \max_q |G(q)|$ then we could be sure that

$$|G^*(x)| \leq m(|x|+1).$$

2.4 Moore Reduction These Moore transformations allow us to place an ordering on sets. Let A be a set of strings over Σ and let B be a set of strings over Ω. We say that

$$A \leq^{\text{Moore}} B$$

if there is a Moore transformation T which for all strings x in Σ^* produces a string $T(x)$ in Ω^* so that

$$x \in A \quad \text{iff} \quad T(x) \in B.$$

This, of course, is a many-one reduction [4] with very strong restrictions on the allowed transformations.

We may note that $HALT$, the halting set, is $\mathcal{RE}$-complete with respect to $\leq^{\text{Moore}}$ since for every $\mathcal{RE}$ set B, there is a Turing acceptor M_B and

$$x \in B \quad \text{iff} \quad M_B : x \in HALT$$

and $M_B : x$ can be computed from x by a Moore machine which outputs from its initial state the string representing the Turing code for M_B followed by the separator : and then computes the identity transformation on x. The identity transformation is computed by a Moore machine with one state for each symbol of Σ. On each input symbol, this machine transitions to the state associated with the symbol and then outputs the symbol.

Moore reductions are probably not of much use for bounded versions of the halting problem, because the necessary transformations require the calculation of a bound $p(|x|)$ as well as M_B and x, and a finite state machine cannot even compute an operation as simple as multiplication.

3 Complete Sets A finite state machine has a YES/NO cycle if there are two states q_1 and q_2, so that

q_1 is an accepting state and q_2 is a rejecting state,
there are two input sequences x_{12} and x_{21},
so that $F(q_1, x_{12}) = q_2$ and $F(q_2, x_{21}) = q_1$,
and q_1 and q_2 are reachable from the initial state.

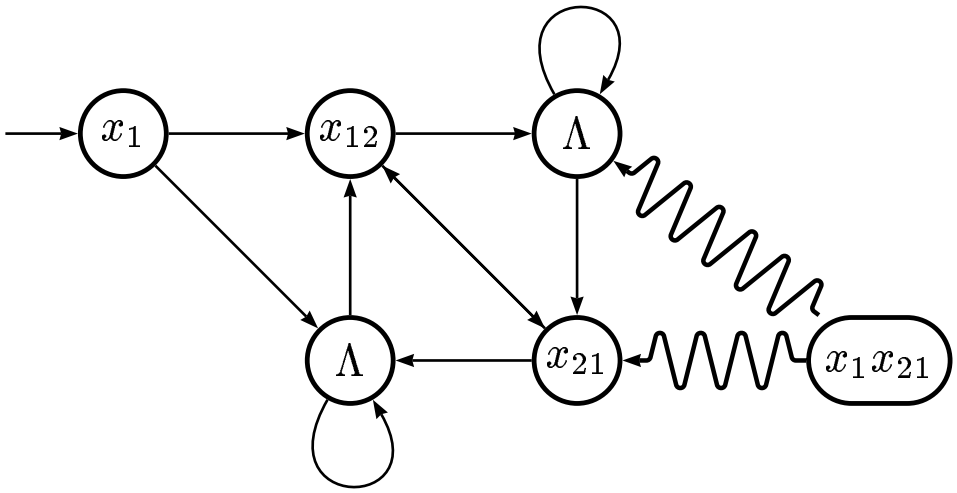
Theorem 1. *A set is Regular-complete with respect to* $\leq^{\text{Moore}}$ *if and only if the set has a finite state recognizer with a YES/NO cycle.*

Proof. **{IF}** Let A be a set with a recognizer M_A with a YES/NO cycle. Let B be any regular set. We construct $f(x)$ which maps strings in B's alphabet to strings in A's alphabet.
If $\Lambda \in B$ then $f(\Lambda) = x_1$ where $F_A(q_0, x_1) = q_1$
else $f(\Lambda) = x_1 x_{12}$ where $F_A(q_0, x_1 x_{12}) = q_2$.
Now assume that $f(w)$ has been defined and consider $f(ws)$ where s is a single symbol. There are four cases

$$\begin{array}{llr} w \in B & ws \in B & f(ws) = f(w)\Lambda = f(w) \\ w \in B & ws \notin B & f(ws) = f(w)x_{12} \\ w \notin B & ws \in B & f(ws) = f(w)x_{21} \\ w \notin B & ws \notin B & f(ws) = f(w)\Lambda = f(w). \end{array}$$

This maintains $x \in B$ iff $f(x) \in A$ because each of the f sequences takes M_A to one of the two state q_1 or q_2 and the inductive definition of f maintains this property. Further this f is computable by a Moore finite state machine. (See the state digram below.) □

The following is a schematic diagram of the transforming Moore machine.



This 5 "state" machine (excluding the state labeled $x_1 x_{21}$) is the transforming machine if $\Lambda \in B$. If $\Lambda \notin B$, the machine with the state labeled x_1 omitted and the $x_1 x_{21}$ state as the initial state transforms B to A. This transformer operates following the 4 cases in the above proof. Since, by assumption, $\Lambda \in B$, the transformer ouputs x_1 on the null input. This output string causes M_A to go to an accepting state q_1. If M_A is in q_1 as a result of the previous output string, then if the next character continues an input string which is in B, the transformer goes to the state on the left labeled Λ and the output string causes no further transition in M_A which stays in the accepting state q_1. If the next input character yields a string not in B, then the transformer goes to the state labeled x_{12} and this output causes M_A to transition to a rejecting state q_2. The cases when M_A is in a rejecting state are handled in a similar manner. If Λ is not in B, then the transformer starts in the state

labeled x_1x_{12} which causes M_A to go to the rejecting state q_2. Subsequent input characters are handled in a manner similar to the behavior when the transformer started in the state labeled x_1.

This is only a schematic picture. The actual transforming machine needs to keep track of the states of M_B to determine if an input symbol causes M_B to transition to an accepting or rejecting state. So each of the 4 middle "states" in the diagram should actually contain a full copy of M_B. E.g. a "state" x_{12} should be, for example, (x_{12}, q_i^B) and on a input symbol s there should be a transition to $(\hat{x}, F_B(q_i^B, s))$ where $\hat{x}$ is a "state" in the diagram. The actual transforming machine would have $4|Q_B| + 1$ states.

Proof. **{ONLY IF}** Conversely, assume that B has a YES/NO cycle, we show that A also has a YES/NO cycle. Then there is a w so that

$$wx_{12} \in B \text{ and } wx_{12}x_{21} \notin B.$$

In fact,

$$w(x_{12}x_{21})^i x_{12} \in B \text{ and } w(x_{12}x_{21})^{i+1} \notin B.$$

But, if $B \leq^{\text{Moore}} A$ then

$$f(w(x_{12}x_{21})^i x_{12}) \in A \text{ and } f(w(x_{12}x_{21})^{i+1}) \notin A.$$

Since the transforming machine has a finite number of states $f(w(x_{12}x_{21})^{i+1})$ must form an eventually periodic sequence $w_0 y^j$. Further, the effect of this sequence on M_A is a state sequence with an infinite number of occurrences of a rejecting state, say q_2. Similarly, $f(w(x_{12}x_{21})^i x_{12})$ must cause a state sequence in M_A with an infinite number of occurrences of an accepting state, say q_1. So the sequence of strings

$$w(x_{12}x_{21}), w(x_{12}x_{21})x_{12}, w(x_{12}x_{21})^2, w(x_{12}x_{21})^2 x_{12},$$
$$\ldots, w(x_{12}x_{21})^i, w(x_{12}x_{21})^i x_{12}, \ldots$$

must be transformed into a sequence of strings which causes M_A to enter q_1 an infinite number of times and to enter q_2 an infinite number of times. So in this state sequence q_1 must occur before some occurrence of q_2 and also after some occurrence of q_2. Hence there is a path from q_1 to q_2 and also a path from q_2 to q_1 and this gives a YES/NO cycle in M_A. □

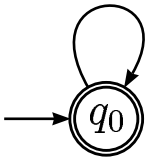
Corollary 1. *A regular set A is Regular-complete with respect to $\leq^{\text{Moore}}$ if and only if there is an infinite word w_∞ (an infinite eventually periodic word p_∞) with an infinite number of prefixes in A and an infinite number of prefixes not in A.*

Proof. The YES/NO cycle in M_A can be used to construct p_∞. Conversely assume that w_∞ exists. Since an infinite number of prefixes of w_∞ are in A, an infinite number of these prefixes cause M_A, which has a finite number of states, to be an accepting state q_1 and an infinite number of these prefixes cause M_A to be a rejecting state q_2. Consider the sequence of M_A's states caused by w_∞. Since the number of q_1's in this sequence is infinite and the number of q_2's is also infinite, there must be some occurrence of q_1 followed eventually by an occurrence of q_2 which is followed eventually by an occurrence of q_1. So there will be some subsequence of w_∞ which causes M_A to transition from q_1 to q_2 and some subsequence of w_∞ which causes M_A to transition from q_2 to q_1, and the YES/NO cycle is established. □

4 Hierarchy

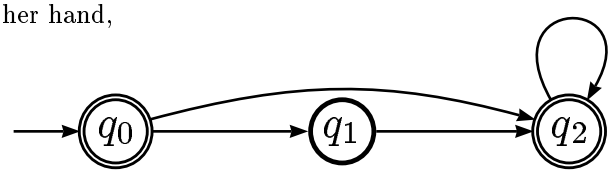
4.1 Alternating Chains Let M_A be a finite state recognizer for a set A. We can limit the states of M_A to those which are reachable from the initial state q_0 without effecting the recognition properties. M_A will have two types of states – accepting and rejecting. For convenience, we'll say that two states have **different parity** if one is an accepting state and the other is a rejecting state. Similarly, two states have the **same parity** if they are both accepting or both rejecting. We can consider the state diagram of M_A as a directed graph. In the usual way, a sequence of states forms a path if there is a transition (an arrow) to each state from its predecessor in the sequence.

Since we saw in Section 3 that recognizers with YES/NO cycles correspond to complete sets, we here want to consider recognizers without such cycles. Such a recognizer has an **alternating chain of length** K iff there is a path M_A in which the parity changes K times. For example,



has an alternating chain of length 0, since $\langle q_0 \rangle$ is a path with no parity changes.

On the other hand,



has an alternating chain of length 0 since $\langle q_0, q_2 \rangle$ forms such a path. This machine also has an altenating path of length 2, $\langle q_0, q_1, q_2 \rangle$, and an alternating path of length 1, $\langle q_1, q_2 \rangle$.

We say that M_A has K alternations, if M_A has an alternating chain of length K and no longer alternating chain. So, for example, the two above machines have respectively $K = 0$ and $K = 2$ alternations.

It is obvious that to find K we only have to consider paths which start in the initial state. Because, if there were a sequence $\langle\, q_{i1},\, q_{i2},\, \ldots,\, q_{ir}\,\rangle$ with alternating length K, then since we only care about reachable states, there is a path from q_0 to q_{i1} and the alternating length of $\langle\, q_0,\, \ldots,\, q_{i1},\, q_{i2},\, \ldots,\, q_{ir}\,\rangle$ is at least K.

The parity of the maximum alternating chain depends on the parity of the initial state q_0. We'll say that the maximum alternating chain has length K_+ if q_0 is accepting, and length K_- if q_0 is rejecting.

For recognizers with YES/NO cycles we can make a (non-simple) path which repetitively goes over the YES/NO cycle, and thus produces an alternating path with arbitarily large length. Reasonably, we set $K = \infty$ for such recognizers. Here, we do not have to make a distinction between sequences which start in an accepting state and those which start in a rejecting state, and so there will only be unsigned ∞.

4.2 Hierarchy Theorem

Theorem 2. *Let K_B be the length of the longest alternating chain in the finite automaton M_B which recognizes B.*

(a) If $K_A < K_B$ then $A \leq^{\text{Moore}} B$ and $B \not\leq^{\text{Moore}} A$.

(b) If $K_B = \infty$ then B is a complete set for regular languages with respect to $\leq^{\text{Moore}}$.

(c) If $K_A = K_B$ and if the longest alternating chains in M_A and M_B start in states with the same parity then $A \leq^{\text{Moore}} B$ and $B \leq^{\text{Moore}} A$ but if these alternating chains start in states of different parity then $A \not\leq^{\text{Moore}} B$ and $B \not\leq^{\text{Moore}} A$.

(d) The degree diagram looks like:

$$\begin{array}{ccccc} K=0_- & \longrightarrow & K=1_- & \longrightarrow \ldots & \searrow \\ & \nearrow\!\!\searrow & & \nearrow\!\!\searrow & K=\infty \\ K=0_+ & \longrightarrow & K=1_+ & \longrightarrow \ldots & \nearrow \end{array}$$

Proof. (a) If M_A has an alternating chain of length K_A then there is an input word w which takes M_A through all the states which make up this chain. Some of the prefixes of w, say

$$p_0, p_1, \ldots, p_K$$

take M_A alternately to accepting and rejecting states. If $A \leq^{\text{Moore}} B$ then there is a finite state function f, so that

$$f(p_0), f(p_1), \ldots, f(p_K)$$

take M_B alternately to accepting and rejecting states. But since this is a finite state mapping, for each i, $f(p_i)$ is a prefix of $f(p_{i+1})$ and so $f(w)$ will take M_B through an alternating chain of length K_A. So if $A \leq^{\text{Moore}} B$ then $K_A \leq K_B$. If $K_A \leq K_B$ and the parities of the initial states doesn't match, the K_A chain would induce a $K_A + 1$ chain in M_B. So, if $K_A \leq K_B$ and $A \leq^{\text{Moore}} B$, then the parity of K_A is the same as the parity of K_B .

If $K_A < K_B$, or $K_A = K_B$ with identical parities, we will construct f which reduces A to B. Let the sequence $w_0, w_1, \ldots, w_K$ be the strings which take M_B to successive alternating states on its longest alternating chain. Define $f(\Lambda) = w_0$ or $f(\Lambda) = w_0 w_1$ so that the parity of the initial state for M_A matches the parity of $F_B(q_0, w_0)$ or $F_B(q_0, w_0 w_1)$. Now assume that $f(w)$ has been defined and consider $f(ws)$ where s is a single symbol. There are four cases

$$\begin{array}{llr} w \in A & ws \in A & f(ws) = f(w)\Lambda = f(w) \\ w \in A & ws \notin A & f(ws) = f(w)w_{i+1} \\ w \notin A & ws \in A & f(ws) = f(w)w_{i+1} \\ w \notin A & ws \notin A & f(ws) = f(w)\Lambda = f(w). \end{array}$$

Here w_{i+1} is the next string in the above sequence which takes M_B to a state of the other parity. This reducing function f always keeps M_B on its longest alternating chain. Since any chain in A has at most K_A alternations, this is a valid reduction as long as $K_A < K_B$, or $K_A = K_B$ with identical parities.

(b) This is just a re-statement of Theorem 1.

(c) The (a) construction shows that if $K_A = K_B$ and if the parity of K_A is the same as the parity of K_b then $A \leq^{\text{Moore}} B$ and $B \leq^{\text{Moore}} A$. But if A and B have opposite parities then, as in the above proof, prefixes of a word in one set must map to prefixes of a word in the other set, and for a valid reduction the number of alternations in the second word must be as great as the number of alternations in the first word. Unfortunately, because the parities differ, the second word is one alternation short.

(d) The degree diagram is simply a graphical re-statement of the theorem.
□

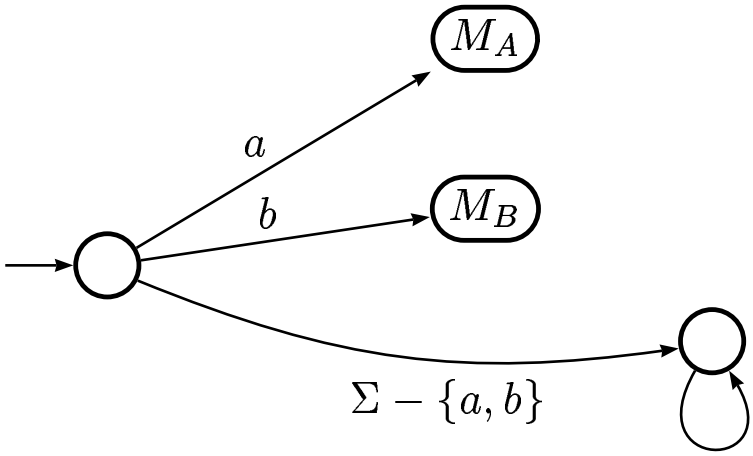
4.3 Least Upper Bounds A set C is the least upper bound (**lub**) with respect to $\leq$ for the sets A and B, if $A \leq C$ and $B \leq C$ and for all D such that $A \leq D$ and $B \leq D$, then $C \leq D$.

The usual **lub** construction [4] uses the symbols a and b, and builds

$$C = \{ z \mid z = ax \text{ with } x \in A \text{ or } z = by \text{ with } y \in B \}.$$

To show that $A \leq^{\text{Moore}} C$ and $B \leq^{\text{Moore}} C$, we need to show that this construction allows a finite state reduction. But $f_A(x) = ax$ is a Moore mapping defined by $f_A(\Lambda) = a$ and $f_A(ws) = f_A(w)s$. Since there is a similar mapping for B, C is an upper bound on A and B, but it is not clear that C is a least upper bound. In fact C is often not an **lub**.

A picture of the recognizer for C is



We can see that a rejecting initial state has been added to the machines M_A and M_B. Let's assume that $K_A > K_B$ then $B \leq^{\text{Moore}} A$. So if C is an **lub** then $C \leq^{\text{Moore}} A$ and also $K_C = K_A$. By the Hierarchy Theorem this will occur exactly when M_A has $-$ parity. If $K_A = K_B$ and they both have $-$ parity, K_C will equal K_A and K_B and C will be an **lub**. If $K_A = K_B$ and they both have $+$ parity, then $K_C = K_A + 1$ and C will not be an **lub**. Finally, if $K_A = K_B$ and they have mixed parity, $K_C = K_A + 1$ and C will not be an **lub** because, in this case, no **lub** can exist, i.e. A and B have upper bounds in $(K_A + 1)_+$ and in $(K_A + 1)_-$ but neither of these upper bounds is reducible to the other.

Corollary 2. *If the* **lub** *for A and B exists, then the* **lub** *is one of A and B.*

4.4 Calculation of Classification Let's assume that we have a finite state recognizer M_B for the set B. We want to find B's position in the hierarchy.

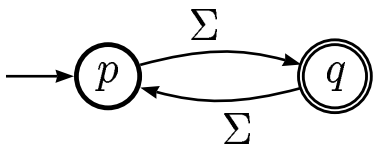
It is easy to determine if M_B has a YES/NO cycle. For example, a depth first search from each state could determine whether or not there is a YES/NO cycle. At most, this

would take $O(nE)$ or $O(n^3)$ time where n is the number of states and E is the number of edges in M_B. Of course, more efficient algorithms are possible.

If there is no YES/NO cycle, we need to determine the length of the longest alternating chain. At first glance, this may seem difficult because it seems to be the notorious $\mathcal{NP}$-hard longest path problem. But it isn't, because we can restrict the search to a directed acyclic graph (a DAG). Consider any strongly connected component of M_B's digraph. Since M_B has no YES/NO cycle, every state in a component must have the same parity. Hence, the search for the longest alternating chain can be done on a DAG whose nodes are the strongly connected components of M_B. As is well known [6] the longest path in a DAG can be found via dynamic programming. A slight variation on this technique can find the longest alternating chain in time $O(n^3)$.

5 EXAMPLES

5.1 Complete Sets Extremely simple looking finite automata can represent complete sets. For example, the following diagram shows an automaton which accepts **ODD** the set of all odd length strings.



This represents a complete set because it has a YES/NO cycle.

5.2 Minimal Sets The lowest levels of the hierarchy contain only the two trivial sets $\emptyset$ and Σ^*. Specifically, $\emptyset$ is the only set with $K = 0_-$ and Σ^* is the only set with $K = 0_+$. Obviously, either of these sets can be reduced to any nontrivial set by finite state mappings which handle the null string appropriately and then map each subsequent character to the null string. Of course, no nontrivial set can be reduced to either of these trivial sets, and neither trivial set can be reduced to the other.

5.3 Finite/coFinite and Definite Events Finite/coFinite is a simple subclass of the regular events. It is the smallest Boolean Algebra which contains the finite sets, or equivalently the smallest concatenation closed Boolean algebra which contains the unit sets.

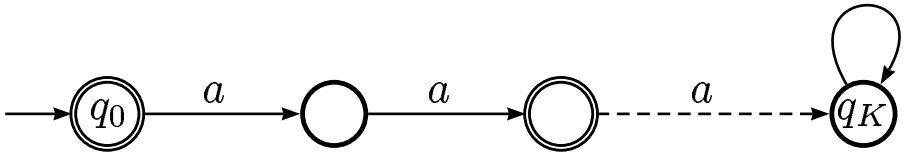
The Definite Events are those sets which can be reconized by neural nets without circles (or feedback loops) in their connections.[7] This class is often confused with Finite/coFinite because each Definite Event can be expressed in the form

$$\Sigma^* \; F$$

where F represents a Finite or coFinite set. In spite of this close relationship,

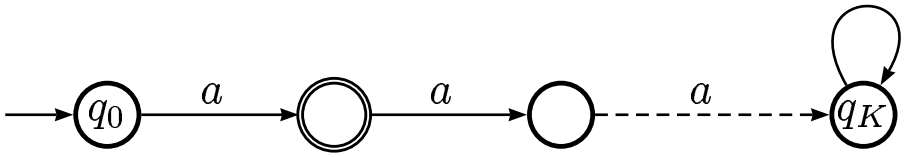
(a) sets in Finite/coFinite occur at every finite level of the Moore Hierarchy,
(b) no sets in Finite/coFinite are Moore complete,
(c) each nontrivial Definite Event is Moore complete.

(a) For any K, consider the following machine with the convention that any missing arrow goes to q_K and the dotted arrow indicates a sequence of alternating parity states.



Clearly, this machine accepts $\{\Lambda, aa, a^4, \ldots, a^{K-1}\}$ and nothing else. This example shows that the degrees with $K = \text{Odd}_+$ contain finite sets.

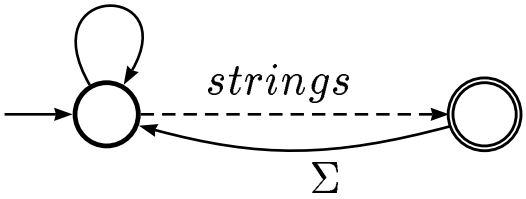
Similarly,



shows the degrees with $K = \text{Even}_-$ contain finite sets. Similar examples show that $K = \text{Even}_+$ and $K = \text{Odd}_-$ contain co-finite sets. Note that $K = \text{Even}_+$ and $K = \text{Odd}_-$ cannot contain finite sets, and $K = \text{Even}_-$ and $K = \text{Odd}_+$ cannot contain co-finite sets.

(b) Since a complete set has a YES/NO cycle, the accepted set must be infinite and the rejected set must be infinite, and so the complete set is neither finite nor co-finite.

(c) Let D be a nontrivial Definite Event. Then there is a string x so that $wx \in D$ for all $w \in \Sigma^*$, and similarly, there is a string y so that $wy \notin D$ for all $w \in \Sigma^*$. Consider the infinite sequence $xyxy\ldots$. Obviously all the prefixes of this sequence which end in x are in D, while all the prefixes which end in y are not in D, and so by Corollary 1, D is Regular complete with respect to $\leq^{\text{Moore}}$.

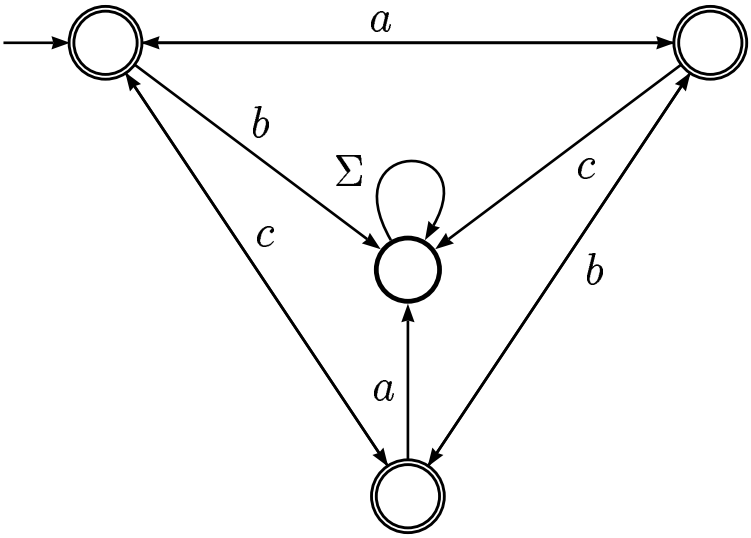


A schematic representation of an automaton which recognizes a nontrivial *definite event*. After accepting a fixed length string, the machine *resets* to consider the next fixed length string. (This is only schematic because the machine does not necessarily have to reset to the initial state, and it does not have to loop around the initial state.) The point of this diagram is that the machines for definite events must have YES/NO cycles.

5.4 Incomplete Sets There are many sets which are infinite and co-infinite and yet are **not** Moore complete. One of my favorite examples is the *Towers of Hanoi* [1] set which is

the set of strings over $\{ a, b, c \}$ which represent the legal configurations of the disks on the towers a, b, and c during the minimal move sequence which takes the disks from tower a to c. I usually take the symbols as being in "reverse" order in the sense that the first character of the input represents the tower on which the largest disk is located. For example, the string $abbb$, says the largest disk is on tower a and the other 3 disks are on tower b. This $abbb$ represents a legal configuration in moving 4 disks from tower a to tower c. On the other hand $baaa$ says that the largest disk is on tower b and the other 3 disks are on tower a and this is not legal configuration.

This is a recognizer for the Towers of Hanoi language:



Notice that this is an infinite and co-infinite language. For each $n \geq 0$, there are 3^n strings, but only 2^n of these strings are in the languge. For this machine $K = 1_+$, so this language appears at a very low level in the hierarchy.

6 CONCLUSION We have shown that a "reasonable" notion of reduction can be defined for the Regular sets. We've called this *Moore* reduction since it uses a Moore finite state machine to compute the reducing function. We've shown that not all Regular sets are equivalent under $\leq^{\text{Moore}}$. Specifically, we let K_A be the length of the longest alternating chain in the minimal recognizer for A, and showed that if $K_A > K_B$, then $A \not\leq^{\text{Moore}} B$, but $B \leq^{\text{Moore}} A$. By considering the parity (whether or not the initial state of the recognizer is accepting), we were able to establish a bi-hierarchy of $\leq^{\text{Moore}}$ degrees. We showed that sets whose recogniers contained a YES/NO cycle are $\leq^{\text{Moore}}$–complete.

A number of features of this reduction seem strange. For example, the **lub** construction fails, and some seemingly simple sets like **ODD** are complete sets, while other seemingly more complicated sets are not complete. We'll leave it to others to decide whether this notion of reduction is useful, but at least it serves as a basis for several counterexamples.

References

[1] P. Cull and E.F. Ecklund Jr. Towers of hanoi and analysis of algorithms. *American Mathematical Monthly*, 92:407–420, 1985.

[2] P. Dunne. *The Complexity of Boolean Networks.* Academic Press, London, 1988.

[3] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.

[4] Hartley Rogers Jr. *Theory of Recursive Functions and Effective Computability.* MIT Press, Cambridge, MA, 1987.

[5] Richard E. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM (JACM)*, 22:155–171, 1975.

[6] E. Lawler. *Combinatorial Optimization: networks and matroids.* Holt, Rinehart and Winston,, New York, NY, 1976.

[7] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulleti of Mathematical Biophysics*, 5:18–27, 1943.

[8] R. McNaughton and S. Papert. *Counter-Free Automata.* MIT Press, Cambridge, MA, 1971.

[9] E. F. Moore. Gedanken experiments on sequential machines. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 129–153. Princeton Univ. Press, Princeton, NJ, 1956.

[10] G. E. Revesz. *Introduction to Formal Languages.* McGraw-Hill, New York, NY, 1983.

[11] I. Wegener. *The Complexity of Boolean Functions.* Teubner, Stuttgart, 1987.

Computer Science Dept.
Oregon State University
Corvallis, OR 97331 USA

pc@cs.orst.edu