WPI-CS-TR-93-11 December 1993

Self-Modifying Finite Automata

by

Roy S. Rubinstein
and
John N. Shutt

Computer Science
Technical Report
Series

WORCESTER POLYTECHNIC INSTITUTE

Computer Science Department
100 Institute Road, Worcester, Massachusetts 01609-2280

Self-Modifying Finite Automata

Roy S. Rubinstein & John N. Shutt
roy@cs.wpi.edu jshutt@cs.wpi.edu
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA 01609

December 1993

Abstract

We present here a new model of computation: the Self-Modifying Finite Automaton (SMFA). This is similar to a standard finite automaton, but changes to the machine are allowed during a computation. It is shown here that a weak form of this model has the power to recognize an important class of context-free languages, the metalinear languages, as well as some significant non-context-free languages. Less restricted forms of SMFA's may accept even more.

1 Introduction

Many abstract models of computation have been devised over the years, including finite automata, pushdown automata, linear-bounded automata, and Turing Machines. This technical report presents a new model: the Self-Modifying Finite Automaton (SMFA). SMFA's are similar to standard finite automata, but changes to a machine are allowed during a computation. While retaining much of the simplicity of finite automata, SMFA's have greater power and can recognize many non-regular and even non-context-free languages.

This paper shows that the classes of linear and metalinear context-free languages are accepted by a weak form of SMFA's, and includes a procedure to construct machines from the appropriate grammars. A subclass of the non-context-free context-sensitive languages is also shown to be accepted by this model, along with the machine construction. Discussion of higher-order SMFA's and their power is also presented.

Although this report is preliminary in nature, lacking formal definitions and answering only a few of the questions it poses, it provides the basic ideas and framework for much future work.

2 Introduction to SMFA's

A self-modifying finite automaton (SMFA) is a (nondeterministic) finite automaton with the additional property of being able to modify itself during a transition from one state to another. Depending on the type of SMFA, the modifications may include adding states, deleting states, adding transitions, and deleting transitions. Modification of the input alphabet is not allowed, nor is changing the set of final states. The modification associated with a transition is called a *modification action* and is part of the transition function. Clearly the power of an SMFA is dependent on the power of the transition functions. If they are arbitrarily powerful, then arbitrarily hard sets, even nonrecursive sets, may be accepted. As such, restrictions will be placed on the transition functions to create different types of SMFA's.

The transition function and set of states before any computation (and modification) are known as the *predefined transition function* and the *set of predefined states*, respectively.

An SMFA may be represented as a quintuple $(Q_0, \Sigma, \delta_0, q_0, F)$, where Q_0 is the set of predefined states, Σ is the input alphabet, δ_0 is the predefined transition function, $q_0 \in Q_0$ is the start state, and $F \subseteq Q_0$ is the set of final states.

While the description to be given in this report is useful, it is rather informal. Formal definitions of SMFA's, including the varieties explained below and definitions of computations, are currently being worked on. Several have been devised, but none so far seem as elegant as they should be. Development work is continuing on this, and formal definitions are expected in the very near future. As such, only informal, though descriptive, definitions are given here.

SMFA's can be drawn with transition diagrams similar to those for standard finite state automata, as a labeled directed graph. Only the predefined states and transition function are drawn, and the labels on the arcs are of the form "a/modification action", where $a \in (\Sigma \cup \{\lambda\})$. The notation of the modification action depends on the type of SMFA and modification action being represented. As the computation progresses, the states and transition function may be appropriately modified.

A *zeroth-order SMFA* is an SMFA where the set of states and the transition function remain constant throughout the computation. Another way to put this is that none of the transitions have any modification actions. Zeroth-order SMFA's are clearly equivalent to standard finite state automata.

A *single-register first-order SMFA* is an SMFA where on any transition at most one transition may be added and no transitions may be deleted. A new transition may not have any modification action associated with it, and the only states that may be addressed "by name" are those in the predefined set of states. In the transition of a modification action a state may be referred to as *new*, meaning that a new state is added and referring to that new state. If both states named in a new transition are *new*, they refer to the same new state. Referring to a state as *old* in a transition is a reference to the last *new* state created (by a previous transition).

The "single-register" refers to the fact that at most one previously created state may be referenced, the most recently created one. An *r-register first-order SMFA* allows reference to r previously created states. These are indicated by subscripts on *new* and *old* in the range of 1 to r. A 0-register SMFA (i.e. one that can only create transitions between predefined states) can be readily seen to be no more powerful than a standard finite automaton. For the remainder of this and the following section, unless otherwise specified, all SMFA's will be first-order single-register SMFA's.

So (single-register) first-order SMFA's are very limited, as they allow only one new transition to be added at a time, and that one can have no modification action attached to it. There is also only one register allowed. In addition, these SMFA's are limited by not having the power to delete transitions.

In spite of these limitations, the class of first-order SMFA's includes an important subclass of the context-free languages and also some non-context-free languages.

3 First-Order SMFA Results

It will be shown that the linear and metalinear context-free languages are all accepted by first-order SMFA's. First, a review of some definitions is in order.

Definition 1 *A context-free grammar $G = (V, \Sigma, P, S)$ is* linear *if each production in P is of the form $A \rightarrow uBv$ or $A \rightarrow u$, where $A, B \in V$ and $u, v \in \Sigma^*$.*

A context-free language is linear *if it is generated by a linear context-free grammar.*

In other words, a linear context-free grammar is a context-free grammar with at most one nonterminal on the right hand side of each production. (In a context-free grammar definition as above, V is the set of nonterminals, Σ is the set of terminals, P is the set of productions, and $S \in V$ is the start symbol.)

Definition 2 *A context-free grammar $G = (V, \Sigma, P, S)$ is* metalinear *if each production in P is of the form $S \rightarrow A_1 A_2 ... A_n$, $A \rightarrow uBv$ or $A \rightarrow u$, where each $A_i, B \in V - \{S\}$, $A \in V$, and $u, v \in \Sigma^*$.*

A context-free language is metalinear *if it is generated by a metalinear context-free grammar.*

So every metalinear language is a finite union of concatenations of linear context-free languages.

Theorem 1 *Every linear context-free language is accepted by some first-order SMFA.*

Proof Let L be generated by a linear context-free grammar $G = (V, \Sigma, P, S)$. We may assume without loss of generality that each production in P is of the form $S \rightarrow A$, $A \rightarrow aBb$ or $A \rightarrow a$, where S is the start symbol, $A, B \in V - \{S\}$ and $a, b \in (\Sigma \cup \{\lambda\})$, as it is easy to build an equivalent grammar of that form.

We will construct a first-order SMFA $M = (Q_0, \Sigma, \delta_0, S, \{q_f\})$ to accept L as follows.
$Q_0 = V \cup \{q_x, q_f\}$, where q_x and q_f are not in V.
δ_0 consists of exactly the following transitions:

- For each production in P of the form $S \rightarrow A$, where S is the start symbol and $A \in V - \{S\}$, add an arc from S to A labeled "λ/add *new* $\xrightarrow{\lambda} q_f$".

$S \rightarrow A$

$A \rightarrow aAb \mid \lambda$

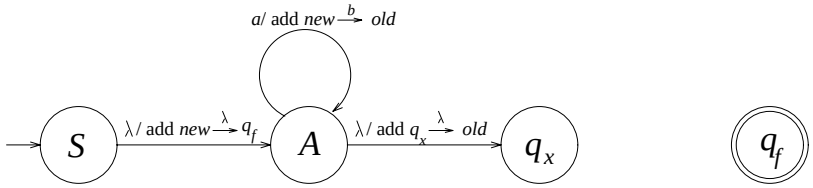

Figure 1: Grammar and SMFA for $\{a^n b^n \mid n \geq 0\}$

$S \rightarrow A$

$A \rightarrow aAa \mid bAb \mid \lambda$

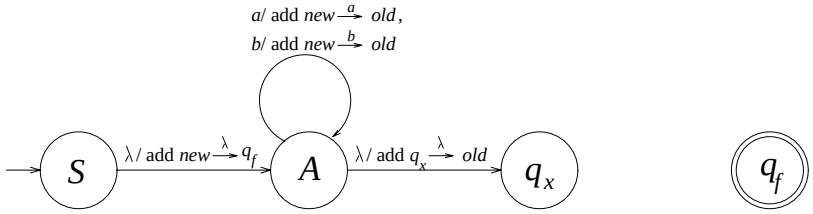

Figure 2: Grammar and SMFA for $\{ww^R \mid w \in \{a,b\}^*\}$

- For each production in P of the form $A \rightarrow aBb$, where $A, B \in V - \{S\}$ and $a, b \in (\Sigma \cup \{\lambda\})$, add an arc from A to B labeled "a/add $new \xrightarrow{b} old$".
- For each production in P of the form $A \rightarrow a$, where $A \in V - \{S\}$ and $a \in (\Sigma \cup \{\lambda\})$, add an arc from A to q_x labeled "a/add $q_x \xrightarrow{\lambda} old$".

The idea of this is that the grammatical derivation is simulated by the states of the SMFA. As the input is processed matching the terminals before the nonterminals on the right hand sides of the productions, states are created backwards from the final state that can be traversed to the final state only by processing input that matches the appropriate terminals that occur to the right of the nonterminal on the right hand side of the productions. This "stack" of new states will only be accessible from the state q_x, and only after no more grammar productions are to be used. □

Thus, through a very simple mechanism, an important subset of the context-free languages may be recognized. As examples, Figure 1 shows a grammar and a first-order SMFA for $\{a^n b^n \mid n \geq 0\}$, and Figure 2 shows a grammar and a first-order SMFA for $\{ww^R \mid w \in \{a,b\}^*\}$.

Corollary 2 *Every metalinear context-free language is accepted by some first-order SMFA.*

Proof It is easy to see that the class of languages accepted by first-order SMFA's is closed under union and concatenation. Union is done by creating a new start state with λ-transitions to the old start states. Concatenation is done by adding a λ-transition from the final state(s) of the first machine to the start state of the next. □

First-order SMFA's can also recognize a number of non-context-free languages, as the following theorem demonstrates.

Theorem 3 *If L is a regular language, then $\{ww \mid w \in L\}$ is accepted by some first-order SMFA.*

Proof Let L be a regular language, generated by a regular grammar $G = (V, \Sigma, P, S)$, so each production is of the form $A \to aB$, or $A \to a$, or $A \to \lambda$, where $A, B \in V$ and $a \in \Sigma$.

We will construct a first-order SMFA $M = (Q_0, \Sigma, \delta_0, S', \{q_f\})$ to accept $\{ww \mid w \in L\}$ as follows.
First, build the augmented grammar $G' = (V \cup \{S'\}, \Sigma, P \cup \{[S' \to S]\}, S')$, where $S' \notin V$. G' is clearly equivalent to G.
$Q_0 = V \cup \{S', q_x, q_f\}$, where q_x and q_f are not in V.
δ_0 consists of exactly the following transitions:

- Add an arc from S' to S labeled "λ/add $q_x \xrightarrow{\lambda} new$".
- For each production in P of the form $A \to aB$, where $A, B \in V$ and $a \in \Sigma$, add an arc from A to B labeled "a/add $old \xrightarrow{a} new$".
- For each production in P of the form $A \to a$, where $A \in V$ and $a \in (\Sigma \cup \{\lambda\})$, add an arc from A to q_x labeled "a/add $old \xrightarrow{a} q_f$".

This is similar to the previous proof, except instead of building a "stack" of new states, we build a "queue" of new states. □

Figure 3 shows a first-order SMFA for the non-context-free language $\{ww \mid w \in \{a, b\}^*\}$.

It appears that in many cases a first-order SMFA is like a standard finite automaton with either a stack or a queue, but without the full power of a stack or queue. Once

$$S' \rightarrow S$$
$$S \rightarrow aS \mid bS \mid \lambda$$

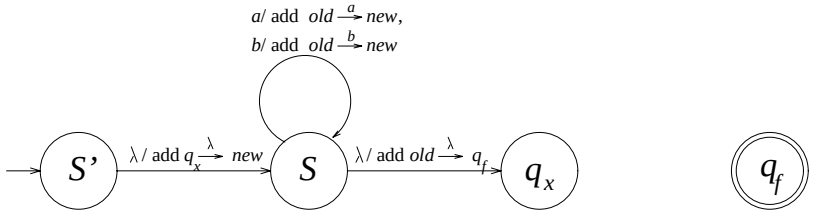

Figure 3: Grammar for $\{a, b\}^*$ and SMFA for $\{ww \mid w \in \{a, b\}^*\}$

popping (or the equivalent queue action) is begun, pushing is no longer permitted until everything has been popped. Once the stack or queue has been emptied, it can be reused, but only finitely many times. This finite constraint is because the set of predefined states is finite and the added transitions cannot have any modification actions.

The information is not really popped, however, but merely traversed. In the above examples it cannot be accessed again, but with a transition back to a state before the new states, it is possible to reuse the new transitions and states. An example of the power of this is seen by following theorem.

Theorem 4 *If L is a regular language, then $\{w^n \mid w \in L$ and $n \geq 0\}$ is accepted by some first-order SMFA.*

Proof This is exactly like the proof of the above theorem, but with q_x and q_f combined, plus an extra transition to accept λ regardless of the regular language. □

Figure 4 shows a grammar for $L(ba^*)$ and a first-order SMFA for the non-context-free language $\{w^n \mid w \in L(ba^*)$ and $n \geq 0\}$.

Similarly, the SMFA for $\{a^n b^n \mid n \geq 0\}$ (Figure 1) can be easily modified to accept $\{a^n b^{kn} \mid n \geq 0$ and $k \geq 0\}$. It should be noted that with only one register, while we can loop through an arbitrary number of times, we have not found a way to set any specific number of times, except for those ≤ 2. For example, we do not know how to construct a single-register first-order SMFA for $\{w^3 \mid w \in \{a, b\}^*\}$.

By allowing a simple form of transition deletion, we can construct a single-register first-order SMFA to accept $\{w^n \mid w \in L\}$ for any $n \geq 0$ and any regular L. This can be accomplished by allowing a transition to be "self-deleting." This means that the transition may be taken at most once, and it deletes itself when used.

$$
\begin{aligned}
S' &\rightarrow S \\
S &\rightarrow bA \\
A &\rightarrow aA \mid \lambda
\end{aligned}
$$

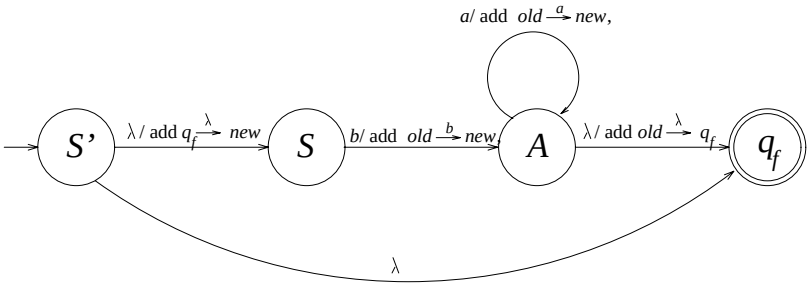

Figure 4: Grammar for $L(ba^*)$ and SMFA for $\{w^n \mid w \in L(ba^*) \text{ and } n \geq 0\}$

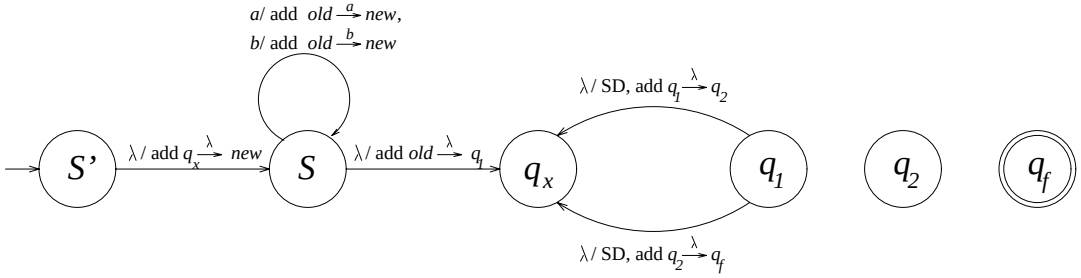

Figure 5: SMFA (with SD) for $\{w^4 \mid w \in \{a, b\}^*\}$

A single-register first-order SMFA to accept $\{w^n \mid w \in L\}$ for a fixed $n > 2$ (we have already seen this for each $n \leq 2$) and a regular language L can be constructed by modifying the SMFA for $\{ww \mid w \in L\}$ (see Theorem 3) as follows. Add $n-2$ additional states (q_1, ..., q_{n-2}) "between" q_x and q_f, and change the reference of q_f to q_1 in the modification action of each transition to q_x. Then add $n-2$ self-deleting λ-transitions from q_1 to q_x, each with a distinct modification action to add a λ-transition from a q_i to q_{i+1}, $1 \leq i \leq n-3$, or from q_{n-2} to q_f.

By having $n-2$ transitions back to q_x from q_1, an additional $n-2$ repetitions of w in the input string is allowed, in addition to the two from the original SMFA. And each of the transitions back to q_x must be taken, as each creates a transition necessary to reach the final state. As an example, Figure 5 shows a machine to accept $\{w^4 \mid w \in \{a, b\}^*\}$, where "SD" indicates a transition is self-deleting.

Single-register first-order SMFA's can also be built to accept non-context-free languages such as $\{a^n b^m c^n d^m \mid m, n \geq 0\}$. This has a bit of the flavor of declaring variables before they are used, indicating that SMFA's may be useful in compiler construction.

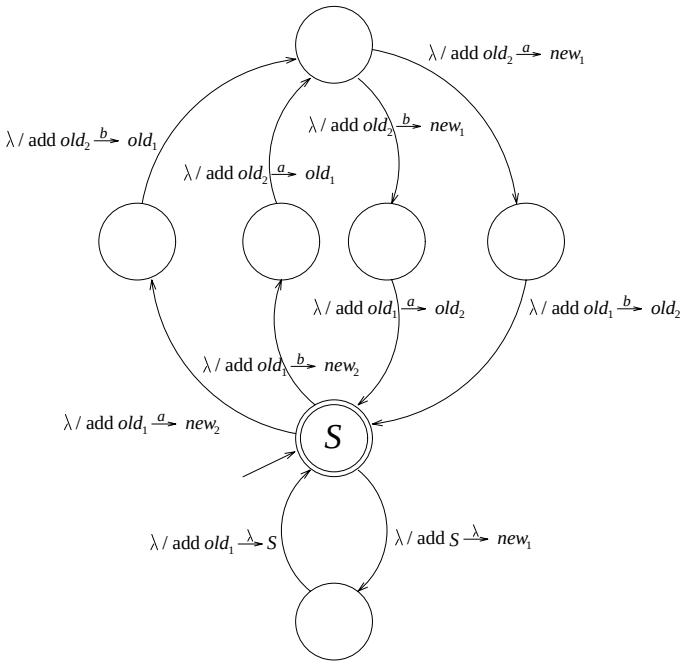

Figure 6: SMFA for same number of a's and b's

4 Multiple-Register SMFA's

With more than one register, it is possible to have more than one stack/queue active at the same time. If the new transitions cannot have any modification action, there would still be a finite maximum number of times each stack/queue could be reused.

The question arises as to whether there are any context-free, non-metalinear languages that are accepted by single-register first-order SMFA's. At this point, we cannot prove that there are any, but we also cannot prove that there are none. We do know, however, that there are context-free, non-metalinear languages that are accepted by 2-register first-order SMFA's. Figure 6 shows a 2-register first-order SMFA that accepts the set of strings over $\{a, b\}$ with the same number of a's and b's, a language known to be context-free but non-metalinear.

The idea is that before reading any of the input, the machine nondeterministically guesses how many a's and b's there will be (actually, the maximum differences between the number of a's and b's in any prefix of the input string) and builds the appropriate machinery. A 2-register first-order SMFA can similarly be built to accept the set of balanced parentheses.

There are also some non-context-free languages accepted by multiple-register SMFA's that appear not to be accepted by single-register SMFA's. It can be easily seen that for any regular language L and any $r \geq 0$, $\{w^{r+1} \mid w \in L\}$ is accepted by some r-register first-order SMFA. This is done as in the proof of Theorem 3, except that when an input symbol is read in, r additional transitions (and states) are created so that that same symbol must be matched in the input r additional times in the correct position. We conjecture (but have not yet proven) that there exist regular L such that for all r, no r-register first-order SMFA accepts $\{w^{r+2} \mid w \in L\}$. This would provide an infinite hierarchy of first-order SMFA's based on the number of registers.

As mentioned above, the class of (single-register or multi-register) first-order SMFA's is closed under union and concatenation. It does not appear to be closed under complement or intersection, but no proof of this is apparent. This needs further investigation.

Interestingly, even though the class is closed under concatenation, it does not appear to be closed under Kleene star, since without a "delete" modification action, there is no way to undo an addition to restore the machine to its initial state. The original machine cannot instead be cloned, as modification actions cannot add transitions with modification actions. There is still no proof of this.

If we allowed a "delete" modification action, we could get a limited version of closure under Kleene star. We could, for example, accept L^* for any linear context-free language L. The construction would be similar to that for accepting L (see Theorem 1), but with a transition from the final state back to the start state. On this transition would be the modification action to delete the transition out of q_x (there must be exactly one), or equivalently to delete the transition into q_f. This prevents taking a path created for a previous string from L.

5 Higher-Order SMFA's

There are a number of ways SMFA's may be made less restrictive. One important way is to allow modification actions to create transitions which themselves have modification actions.

One way to define this is to define a hierarchy of modification actions, where a modification action may create a transition with a modification of a lower order. Definitions are as follows:

Definition 3 *A* zeroth-order modification action *is the empty modification action.* (In other words, the machine stays the same – no transitions or states are added or deleted.)

A $k+1^{st}$-order modification action *allows adding one transition which itself has a* k^{th}*-order modification action.*

A k^{th}-order SMFA *is an SMFA with* k^{th}*-order or lower modification actions.*

A general SMFA *is* $\bigcup_{k\geq 0}$ k^{th}*-order SMFA.*

An unrestricted SMFA *is an SMFA with modification actions of unbounded order.*

So the modification actions on a $k+1^{st}$-order SMFA are the transition labels of a k^{th}-order SMFA. Just how the modification actions would be specified still would need to be defined. The idea of "cloning" a state and/or transition seems like it may be useful as well. As there are as yet no results concerning higher-level SMFA's, it remains to be seen if these definitions are appropriate, and they may need revision.

Additionally, allowing modification actions to delete transitions might do more (as previously discussed), and may even make the model Turing powerful. Allowing multiple modification actions on a single transition creates an additional flavor of SMFA. How these fit in with the other SMFA's in regards to power has yet to be determined. Other variations include prohibiting λ-transitions and nondeterminism. How these variations would fit appropriately into definitions of higher-order SMFA's are as yet undetermined.

6 Future Work

At this point, while there are languages that intuitively seem to be unrecognizable by first-order SMFA's, proofs have been elusive. No proofs have yet been found to show that *any* language that is recognizable by a Turing machine is not recognizable by a single-register first-order SMFA. There appears to be evidence of such languages, but no proof. We conjecture that every language accepted by a first-order SMFA must be context-sensitive, but that has yet to be proven.

The idea of having a machine modify itself may also be applied to other types of machines, such as pushdown automata, linear-bounded automata, and Turing machines. These topics need further investigation.

Finally, this research may spawn new research in the area of complexity theory, as time- and space-bounded versions of these machines may be of interest.

References

[Har78] M. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, Reading, Massachusetts, 1978.

[HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Massachusetts, 1979.

[LP81] H. Lewis and C. Papadimitriou. *Elements of the Theory of Computation.* Prentice Hall, Inc., Englewood Cliffs, NJ, 1981.