An O(n^2) Time Algorithm for Deciding Whether a Regular Language is a Code

Robert McCloskey
Department of Computing Sciences
University of Scranton
mccloske@cs.uofs.edu

Abstract

We describe an algorithm that, given as input a nondeterministic finite automaton A of size n, decides in $\mathrm{O}(n^2)$ time whether the language accepted by A is a code. Let Σ be a finite alphabet. A language $\eta \subseteq \Sigma^*$ is a code if every string in η^* is written uniquely as a concatenation of strings from η, or, equivalently, if η^* is a free submonoid of Σ^* and η is its minimal generating set.

Keywords: algorithm, code, finite automaton, regular language, free monoid

1 Introduction

Informally, a *code* is a set of strings such that any "message" composed from its members is uniquely decipherable. In [10], Sardinas and Patterson characterized codes in a way that gave rise to the earliest algorithm for deciding whether a given finite set of strings is a code. (See also Chapter 5 of [8] or Chapter 4 of [9].) Other algorithms for this problem followed, including those described in [3] and [11].

An algorithm for the more general problem of deciding whether a given *regular* (also called *rational* or *recognizable*) language is a code is given in [2]. This algorithm runs in polynomial time, but its input is assumed to be an unambiguous finite automaton (i.e., one having a transition graph in which, for any string x and any states p and q, there is at most one walk from p to q spelling out x). Thus, if the regular language is given by an ambiguous finite automaton, it must be "disambiguated" before the algorithm can be applied. As pointed out in [5], this construction can result in an exponential explosion in the number of states.

In [5], Head and Weber avoided this pitfall by taking a different approach: they showed that the problem is reducible to that of deciding whether a given nondeterministic finite transducer is single-valued, which was proved in [4] to be solvable in polynomial time.

Here, we show that the algorithm arising directly from [10]—which is applicable only to finite languages—can be generalized to one that is capable of determining whether a given regular language is a code. The input is assumed to be given in the form of (the transition graph of) a (possibly ambiguous) nondeterministic finite automaton (henceforth, NFA) A; the algorithm's running time is $\mathrm{O}(n^2)$, where n is the sum of the numbers of states and transitions in A. The number of symbols in A's alphabet is taken to be a constant.

The paper has three sections following this one. Section 2 covers some concepts and notation pertaining to NFA's and formal languages. In Section 3, we characterize NFA's that accept codes. In the final section, the algorithm is sketched and its asymptotic running time is determined.

2 Notation and terminology

The reader is assumed to be familiar with the notion of a directed graph[1] and with the fundamentals of formal languages (e.g., alphabet, string, language). Prior exposure to the theory of finite automata and regular languages would be very helpful. Generally, our notation is consistent with [6] and [7]. Throughout the paper, Σ denotes a finite alphabet and λ the empty string.

Along with the usual set-theoretic operations on languages (e.g., union, intersection), we use extensively *left quotient*, which, for $\alpha, \beta \subseteq \Sigma^*$ is defined by

$$\alpha \setminus \beta = \{y \mid xy \in \beta \text{ for some } x \in \alpha\}$$

Also central to our work is the notion of an NFA, which is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of *states*, Σ is a finite *alphabet*, $\delta : Q \times (\Sigma \cup \{\lambda\}) \to 2^Q$ is the *transition function*, $q_0 \in Q$ is the *initial state*, and $F \subseteq Q$ is the set of *accepting states*. A is naturally identified with the directed, edge-labeled graph T_A (called its *transition graph*) with node set Q and edge set $E = \{(p, a, s) \in Q \times (\Sigma \cup \{\lambda\}) \times Q \mid s \in \delta(p, a)\}$. (The triple (p, a, s) denotes the edge from p to s labeled a.) From now on, we will abuse the notation by not bothering to distinguish between an NFA and its transition graph.

By extending the domain of δ to $Q \times \Sigma^*$ in the natural way, we get the function $\delta^* : Q \times \Sigma^* \to 2^Q$ satisfying $q \in \delta^*(p, x)$ iff there exists a *walk* in A from p to q whose edge labels (when concatenated) "spell out" the string x. (See [6].) The language accepted by A, denoted $L(A)$, is $\{x \in \Sigma^* \mid \delta^*(q_0, x) \cap F \neq \emptyset\}$, i.e., the set of strings spelled out by walks in A beginning at q_0 and ending in accepting states. The domain of δ^* is extended to $2^Q \times 2^{\Sigma^*}$ by letting

$$\delta^*(P, \eta) = \bigcup_{p \in P, x \in \eta} \delta^*(p, x)$$

If A has only one accepting state and there are no λ-transitions into that state and no transitions of any kind out of that state, A is said to be in *restricted form*. The NFA depicted

[1] Strictly speaking, we use *pseudographs*, which may include edges from a node to itself and/or multiple edges between two nodes.

in Figure 1 is in restricted form and accepts the language given by the regular expression $b(aba + ba)^*b$.

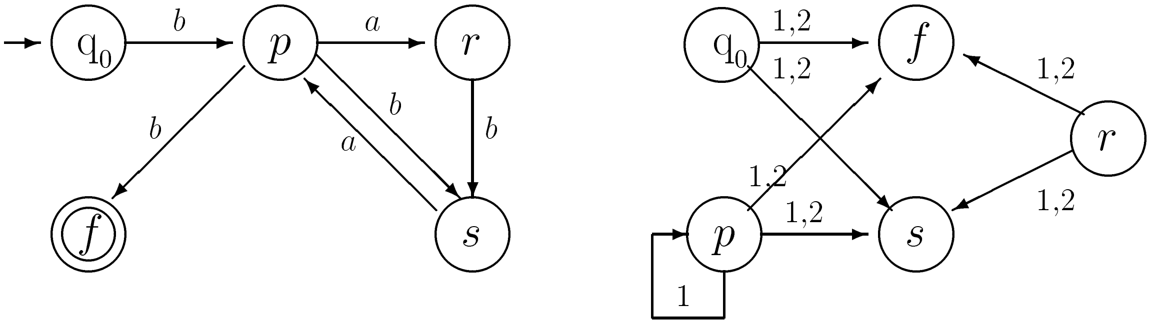

Figure 1: An NFA and its Patterson-Sardinas graph

In the remainder of the paper, we fix $A = (Q, \Sigma, \delta, q_0, F)$. Whenever A is stipulated to be in restricted form, fix $F = \{f\}$.

Where A is understood from context and $p, s \in Q$, we use $L_{p\to s}$ to denote the set of strings $\{x \in \Sigma^* \mid s \in \delta^*(p, x)\}$, i.e., $x \in L_{p\to s}$ iff x is spelled out by some walk in A beginning at p and ending at s. Where $P, S \subseteq Q$, let

$$L_{P\to S} = \{x \in \Sigma^* \mid S \cap \delta^*(P, x) \neq \emptyset\}$$

We define $L_{p\to S}$ and $L_{P\to s}$ similarly. The following two lemmas are obvious:

Lemma 1 *For any $P, P', S, S' \subseteq Q$, if $P \subseteq P'$ and $S \subseteq S'$, then $L_{P\to S} \subseteq L_{P'\to S'}$*

Lemma 2 *For any $P, R, S \subseteq Q$, $L_{P\cup R\longrightarrow S} = L_{P\longrightarrow S} \cup L_{R\longrightarrow S}$*

For our first non-trivial lemma, we have

Lemma 3 *For any $P, R, S, T \subseteq Q$, $L_{P\longrightarrow R}\backslash L_{S\longrightarrow T} = L_{\delta^*(S, L_{P\to R})\to T}$.*

Proof: Let $y \in L_{P\to R} \setminus L_{S\to T}$. By definition of left quotient, there exists some $x \in L_{P\longrightarrow R}$ such that $xy \in L_{S\longrightarrow T}$. It follows that $y \in L_{q\longrightarrow T}$ for some state $q \in \delta^*(S, x)$. But then $y \in L_{q\longrightarrow T} \subseteq L_{\delta^*(S,x)\longrightarrow T} \subseteq L_{\delta^*(S, L_{P\longrightarrow R})\longrightarrow T}$, where the two set inclusions are due to Lemma 1 together with the observation that $\{q\} \subseteq \delta^*(S, x) \subseteq \delta^*(S, L_{P\longrightarrow R})$.

Let $y \in L_{\delta^*(S, L_{P\longrightarrow R})\longrightarrow T}$. For some state $q \in \delta^*(S, L_{P\longrightarrow R})$, $y \in L_{q\longrightarrow T}$. It follows that $x \in L_{S\longrightarrow q}$ for some $x \in L_{P\longrightarrow R}$. From $x \in L_{S\longrightarrow q}$ and $y \in L_{q\longrightarrow T}$, it is evident that $xy \in L_{S\longrightarrow T}$. Now, $\{y\} = \{x\}\backslash\{xy\}$, $x \in L_{P\longrightarrow R}$, and $xy \in L_{S\longrightarrow T}$. Thus, $y \in L_{P\longrightarrow R}\backslash L_{S\longrightarrow T}$. ■

3 Characterizing NFA's that accept codes

Definition 1 *A language $\eta \subseteq \Sigma^*$ is a* code *if, whenever $x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_n$ are in η and satisfy*

$$x_1 x_2 \cdots x_m \;=\; y_1 y_2 \cdots y_n,$$

then $m = n$ and $x_i = y_i$ for all i, $1 \leq i \leq m$.

For example, $\{ab, ba, bb, abbab\}$ is *not* a code, because $ab \cdot ba \cdot ba \cdot bb \cdot ab = abbab \cdot abbab$. We leave it to the reader to verify that the removal of any (one or more) of the strings from this set results in a code.

In [10], codes are characterized according to a sequence of sets, called "segments" (here denoted by subscripted $\mathcal{S}$'s), which are formed by repeated application of the left quotient operation. The details follow.

Definition 2 *For $\eta \subseteq \Sigma^*$, define*

$$\begin{aligned} \mathcal{S}_0(\eta) &= \eta \\ \mathcal{S}_1(\eta) &= \eta \setminus \eta \;-\; \{\lambda\} \\ \mathcal{S}_{i+1}(\eta) &= \eta \setminus \mathcal{S}_i(\eta) \;\cup\; \mathcal{S}_i(\eta) \setminus \eta \qquad (i > 0) \end{aligned}$$

Theorem 1 ([10]) *Let $\eta \subseteq \Sigma^*$. Then η is a code iff $\lambda \notin \eta$ and*

$$\eta \;\cap\; \bigcup_{i>0} \mathcal{S}_i(\eta) \;=\; \emptyset$$

It is clear that, for any $i > 0$, $\eta \,\cap\, \mathcal{S}_i(\eta) = \emptyset$ iff $\lambda \notin \mathcal{S}_{i+1}(\eta)$. From this, we get the following:

Corollary 1 *A language $\eta \subseteq \Sigma^*$ is a code iff*

$$\lambda \notin \bigcup_{i \geq 0} \mathcal{S}_i(\eta)$$

Suppose $\eta \subseteq \Sigma^*$ is finite, and let $x \in \eta$ be a string of maximal length in η. Then, by the properties of left quotient, there can be no string in any segment $\mathcal{S}_i(\eta)$, $i \geq 0$, whose length exceeds that of x. Thus, the union $\cup_{i \geq 0} \mathcal{S}_i(\eta)$ of all segments is a finite set. Moreover, if, for some $k \geq 0$, $\mathcal{S}_k(\eta) \subseteq \cup_{0 \leq i < k} \mathcal{S}_i(\eta)$, then the union of all segments is equal to $\cup_{0 \leq i < k} \mathcal{S}_i(\eta)$. (This statement is true even if η is infinite.) Using these facts together with Theorem 1 (or its corollary), it is easy to devise an algorithm for deciding whether a finite language is a code.

For the purpose of generalizing this algorithm to the case of regular languages, below we define the state sets $\mathcal{Q}_i$ ($i \geq 0$), which, in analogy to the segments $\mathcal{S}_i$ ($i \geq 0$) of Definition 2, serve as the basis of our characterization of NFA's that accept codes.

Definition 3 *For an NFA A, define $\Phi_A : Q \times Q \to 2^Q$ as follows:*

$$\Phi_A(p, s) = \delta^*(s, L_{p\to F})$$

(Where A is clear from context, the subscript is omitted.) The domain of Φ is extended in the natural way to allow either argument to be a set. For example, $\Phi(P, S) = \delta^*(S, L_{P\to F})$. We define $\Phi(p, S)$ and $\Phi(P, s)$ similarly.

We now state two lemmas regarding the above definition; the first follows immediately from it.

Lemma 4 *For $p, s \in Q$, $\Phi(p, s) = \{r \in Q \mid L_{s\to r} \cap L_{p\to F} \neq \emptyset\}$.*

Lemma 5 $L_{P\to F} \setminus L_{S\to F} = L_{\Phi(P,S)\to F}$

Proof: Replace R and T by F in Lemma 3 and use the definition of Φ. ■

Definition 4 *For an NFA A, define*

$$\begin{aligned} \mathcal{Q}_0(A) &= \{q_0\} \\ \mathcal{Q}_1(A) &= \Phi(q_0, q_0) \; - \; F \\ \mathcal{Q}_{i+1}(A) &= \Phi(q_0, \mathcal{Q}_i(A)) \; \cup \; \Phi(\mathcal{Q}_i(A), q_0) \qquad (i > 0) \end{aligned}$$

Lemma 6 *Let $\eta = L(A)$, with A in restricted form. Then, for all $k \geq 0$,*

$$L_{\mathcal{Q}_k(A)\to f} = \mathcal{S}_k(\eta).$$

Proof: We use induction on k. The basis covers $k = 0$ and $k = 1$. For $k = 0$, we have

$$L_{\mathcal{Q}_0(A)\to f} = L_{q_0\to f} = L(A) = \eta = \mathcal{S}_0(\eta)$$

For $k = 1$:

$$\begin{aligned} L_{\mathcal{Q}_1(A)\to f} &= L_{(\Phi(q_0,q_0)-\{f\})\to f} && \text{(defn. of } \mathcal{Q}_1) \\ &= L_{\Phi(q_0,q_0)\to f} \; - \; \{\lambda\} && (L_{f\to f} = \{\lambda\};\; \lambda \in L_{q\to f} \Rightarrow q = f) \\ &= L_{q_0\to f} \setminus L_{q_0\to f} \; - \; \{\lambda\} && \text{(Lemma 5)} \\ &= \eta\setminus\eta \; - \; \{\lambda\} && (L_{q_0\to f} = L(A) = \eta) \\ &= \mathcal{S}_1(\eta) && \text{(defn. of } \mathcal{S}_1) \end{aligned}$$

For the induction step, let $k \geq 1$.

$$\begin{aligned} & L_{\mathcal{Q}_{k+1}(A)\to f} \\ = \; & L_{(\Phi(q_0,\mathcal{Q}_k(A)) \,\cup\, \Phi(\mathcal{Q}_k(A),q_0))\to f} && \text{(defn. of } \mathcal{Q}_{k+1}) \\ = \; & L_{\Phi(q_0,\mathcal{Q}_k(A))\to f} \; \cup \; L_{\Phi(\mathcal{Q}_k(A),q_0)\to f} && \text{(Lemma 2)} \\ = \; & L_{q_0\to f} \setminus L_{\mathcal{Q}_k(A)\to f} \; \cup \; L_{\mathcal{Q}_k(A)\to f} \setminus L_{q_0\to f} && \text{(Lemma 5)} \\ = \; & \eta\setminus\mathcal{S}_k(\eta) \; \cup \; \mathcal{S}_k(\eta)\setminus\eta && \text{(ind. hyp.; } L_{q_0\to f} = L(A) = \eta) \\ = \; & \mathcal{S}_{k+1}(\eta) && \text{(defn. of } \mathcal{S}_{k+1}) \end{aligned}$$ ■

Theorem 2 *For A in restricted form, $L(A)$ is a code iff*

$$f \notin \bigcup_{i \geq 0} \mathcal{Q}_i(A)$$

Proof: It follows from Lemma 6, Corollary 1, and $\lambda \in L_{q \to f}$ iff $q = f$. ■

We now introduce a concept that, although not used directly in the algorithm, aids us in proving its correctness.

Definition 5 *The Patterson-Sardinas graph of A, denoted $\mathcal{PS}(A)$, is the directed graph with node set Q and edge set $E = E_1 \cup E_2$, where*

$$E_1 = \{(p, s) \mid s \in \Phi(p, q_0)\} \text{ and } E_2 = \{(p, s) \mid s \in \Phi(q_0, p)\}^2$$

Figure 1 shows an NFA and its Patterson-Sardinas graph. (Each edge of the latter is labeled by 1 and/or 2 to show from which set, E_1 and/or E_2, it comes.) A *walk* $s_0, s_1, \ldots, s_n$ in $\mathcal{PS}(A)$ is said to be *good* if either $n = 0$ or $s_1 \notin F$. We use $p \overset{k}{\rightsquigarrow} s$ to denote that there is a good walk of length k from p to s in $\mathcal{PS}(A)$.

Lemma 7 *Let A be in restricted form. For all $s \in Q$ and all $k \geq 0$, there is a good walk of length k in $\mathcal{PS}(A)$ from q_0 to s (i.e., $q_0 \overset{k}{\rightsquigarrow} s$) iff $s \in \mathcal{Q}_k(A)$.*

Proof: We use induction on k. The basis covers $k = 0$ and $k = 1$. For $k = 0$, we have

$$q_0 \overset{0}{\rightsquigarrow} s \;\equiv\; s = q_0 \;\equiv\; s \in \mathcal{Q}_0(A)$$

For $k = 1$:

$$\begin{array}{lll}
 & q_0 \overset{1}{\rightsquigarrow} s & \\
\equiv & s \neq f \wedge ((q_0, s) \in E_1 \vee (q_0, s) \in E_2) & \text{(defn. of (good) walk)} \\
\equiv & s \neq f \wedge (s \in \Phi(q_0, q_0) \vee s \in \Phi(q_0, q_0)) & \text{(defn. of } E_1, E_2\text{)} \\
\equiv & s \neq f \wedge s \in \Phi(q_0, q_0) & (P \vee P \equiv P) \\
\equiv & s \in \Phi(q_0, q_0) - \{f\} & (x \neq y \wedge x \in Z \;\equiv\; x \in Z - \{y\}) \\
\equiv & s \in \mathcal{Q}_1(A) & \text{(defn. of } \mathcal{Q}_1\text{)}
\end{array}$$

[2]By Lemma 4, we can also say $E_1 = \{(p, s) \mid L_{q_0 \to s} \cap L_{p \to F} \neq \emptyset\}$ and $E_2 = \{(p, s) \mid L_{q_0 \to F} \cap L_{p \to s} \neq \emptyset\}$.

For the induction step, let $k \geq 1$.

$$
\begin{array}{lll}
& q_0 \overset{k+1}{\rightsquigarrow} s & \\
\equiv & (\exists s')(q_0 \overset{k}{\rightsquigarrow} s' \wedge ((s',s) \in E_1 \vee (s',s) \in E_2)) & \text{(defn. of (good) walk)} \\
\equiv & (\exists s')(s' \in \mathcal{Q}_k(A) \wedge ((s',s) \in E_1 \vee (s',s) \in E_2)) & \text{(ind. hyp.)} \\
\equiv & (\exists s')(s' \in \mathcal{Q}_k(A) \wedge (s \in \Phi(q_0,s') \vee s \in \Phi(s',q_0))) & \text{(defn. of } E_1, E_2) \\
\equiv & (\exists s')((s' \in \mathcal{Q}_k(A) \wedge s \in \Phi(q_0,s')) \vee & \\
& \qquad (s' \in \mathcal{Q}_k(A) \wedge s \in \Phi(s',q_0))) & (\wedge \text{ distributes over } \vee) \\
\equiv & (\exists s')(s' \in \mathcal{Q}_k(A) \wedge s \in \Phi(q_0,s')) \vee & \\
& (\exists s')(s' \in \mathcal{Q}_k(A) \wedge s \in \Phi(s',q_0)) & (\exists \text{ distributes over } \vee) \\
\equiv & s \in \Phi(q_0, \mathcal{Q}_k(A)) \vee s \in \Phi(\mathcal{Q}_k(A), q_0) & \text{(defn. of } \Phi) \\
\equiv & s \in \Phi(q_0, \mathcal{Q}_k(A)) \cup \Phi(\mathcal{Q}_k(A), q_0) & (x \in Y \vee x \in Z \equiv x \in Y \cup Z) \\
\equiv & s \in \mathcal{Q}_{k+1}(A) & \text{(defn. of } \mathcal{Q}_{k+1}) \ \blacksquare
\end{array}
$$

Theorem 3 *Let A be in restricted form. Then $L(A)$ is a code iff there are no good walks from q_0 to f in $\mathcal{PS}(A)$.*

Proof: The statement follows immediately from Lemma 7 and Theorem 2. $\blacksquare$

Consider the NFA A and its Patterson-Sardinas graph $\mathcal{PS}(A)$, both shown in Figure 1. There are no good walks in $\mathcal{PS}(A)$ from q_0 to f; hence, $L(A)$ (i.e., $b(aba+ba)^*b$) is a code. Suppose that we were to construct A' by adding the edge (p,a,f) to A. The graph $\mathcal{PS}(A')$ contains all the edges of $\mathcal{PS}(A)$, plus others, including (q_0, r). Thus, in $\mathcal{PS}(A')$ there is a good walk q_0, r, f, indicating that $L(A')$ (i.e., $b(aba+ba)^*(a+b)$) is not a code.

4 The algorithm

The algorithm is based on Theorem 3. However, it is not necessary to construct $\mathcal{PS}(A)$ in order to determine whether that graph contains a good walk from q_0 to f. (This construction can be performed in $O(n^3)$ time, but, as far as we could determine, no faster.) It suffices, instead, to construct (a variant of) the *direct product* of A (see [6]), which is given by $\langle A \times A \rangle = (Q \times Q, \Sigma, \delta', [q_0, q_0], [f, f])$, where, for $p, q \in Q$,

$$
\begin{array}{rcll}
\delta'([p,q],a) & = & \delta(p,a) \times \delta(q,a) & (a \in \Sigma) \\
\delta'([p,q],\lambda) & = & ((\delta(p,\lambda) \cup \{p\}) \times (\delta(q,\lambda) \cup \{q\})) - \{[p,q]\} &
\end{array}
$$

It is easy to verify (by induction on $|x|$) that $[r,s] \in \delta'^*([p,q],x)$ iff $r \in \delta^*(p,x)$ and $s \in \delta^*(q,x)$. That is, there is a walk in $\langle A \times A \rangle$ from $[p,q]$ to $[r,s]$ spelling out x iff A has walks from p to r and from q to s, both of which spell out x. Comparing this with the definition of the edge set $E_1 \cup E_2$ of $\mathcal{PS}(A)$ and making use of Lemma 4, as well as the symmetry of $\langle A \times A \rangle$, the following becomes evident:

Lemma 8 *Let A be in restricted form, and let $p, s \in Q$. Then*

$$(p,s) \in E_1 \ \text{ iff } \ [s,f] \in \delta'^*([q_0,p],x) \text{ for some } x \in \Sigma^* \ \text{ and}$$

$$(p, s) \in E_2 \quad \text{iff} \quad [f, s] \in \delta'^*([q_0, p], x) \text{ for some } x \in \Sigma^*.$$

That is, $(p, s) \in E_1$ (respectively, E_2) iff there is a walk in $\langle A \times A \rangle$ from $[q_0, p]$ to $[s, f]$ (respectively, $[f, s]$).

Let $\hat{A}$ be the directed graph obtained by taking $\langle A \times A \rangle$ and inserting, for each $s \neq f$, an edge from each of $[s, f]$ and $[f, s]$ to $[q_0, s]$. Refer to a state $[s, f]$ or $[f, s]$, where $s \neq f$, as a *semi-final* state. It follows from A being in restricted form that, in $\langle A \times A \rangle$, any transition out of a semi-final state must be a λ-transition to another semi-final state.

Lemma 9 *Let A be in restricted form. Then there exists a good walk in $\mathcal{PS}(A)$ from q_0 to f iff there exists a walk in $\hat{A}$ from $[q_0, q_0]$ to $[f, f]$ that passes through at least one semi-final state.*

Proof: Let $p_0, p_1, \ldots, p_k$ be a good walk in $\mathcal{PS}(A)$, where $p_0 = q_0$ and $p_k = f$. By definition of good walk, this requires $k > 1$. From Lemma 8 it follows that, for each i, $0 \leq i < k$, there is a walk in $\langle A \times A \rangle$ (and hence in $\hat{A}$) from $[q_0, p_i]$ to either $[p_{i+1}, f]$ or $[f, p_{i+1}]$, the former if $(p_i, p_{i+1}) \in E_1$ and the latter if $(p_i, p_{i+1}) \in E_2$. By the construction of $\hat{A}$, there are edges from both $[p_{i+1}, f]$ and $[f, p_{i+1}]$ to $[q_0, p_{i+1}]$ for all i, $0 \leq i < k - 1$. Thus, there is a walk in $\hat{A}$ that begins at $[q_0, q_0]$, ends at $[f, f]$, and, for each i satisfying $1 \leq i < k$, passes through either $[f, p_i]$ or $[p_i, f]$.

For the converse, let

$$\mathcal{W} = [q_0, q_0], \ldots, [p_1, r_1], \ldots, [p_2, r_2], \ldots\ldots, [p_k, r_k], \ldots, [f, f]$$

be a walk in $\hat{A}$, where $k > 0$ and where the states $[p_i, r_i]$, $1 \leq i \leq k$, are all those on $\mathcal{W}$ that are both semi-final and not immediately followed on $\mathcal{W}$ by another semi-final state. For each i, let s_i be the one among p_i and r_i that is not f. By the construction of $\hat{A}$, the only edge that leaves $[p_i, r_i]$ and enters a state that is not also semi-final goes to $[q_0, s_i]$. Thus,

$$\mathcal{W} = [q_0, q_0], \ldots, [p_1, r_1], [q_0, s_1], \ldots, [p_2, r_2], [q_0, s_2], \ldots\ldots, [p_k, r_k], [q_0, s_k], \ldots, [f, f]$$

Moreover, any walk in $\hat{A}$ that does not include an edge from a semi-final state to a non-semi-final state is also a walk in $\langle A \times A \rangle$. Referring to $\mathcal{W}$, we conclude that in $\langle A \times A \rangle$ there are walks from $[q_0, q_0]$ to $[p_1, r_1]$, from $[q_0, s_i]$ to $[p_{i+1}, r_{i+1}]$ (for each i satisfying $1 \leq i < k$), and from $[q_0, s_k]$ to $[f, f]$. Thus, by Lemma 8, either $(q_0, p_1) \in E_1$ (in the case $r_1 = f$) or $(q_0, r_1) \in E_2$ (in the case $p_1 = f$). Either way, $(q_0, s_1) \in E$. By similar reasoning, for each i satisfying $1 \leq i < k$, either $(s_i, p_{i+1}) \in E_1$ or $(s_i, r_{i+1}) \in E_2$. Either way, $(s_i, s_{i+1}) \in E$. Finally, by Lemma 8 once more, $(s_k, f) \in E$. It follows that $q_0, s_1, s_2, \ldots, s_k, f$ is a walk in $\mathcal{PS}(A)$. Because $k > 0$, it is a good walk. ■

From Theorem 3 and Lemma 9, the following is immediate:

Theorem 4 *Let A be in restricted form. Then $L(A)$ is a code iff there are no walks in $\hat{A}$ from $[q_0, q_0]$ to $[f, f]$ that pass through at least one semi-final state.*

Assuming that each step can be completed in $\mathrm{O}(n^2)$ time, Theorem 4 implies that, given as input an NFA A' of size n, the following algorithm correctly decides, in $\mathrm{O}(n^2)$ time, whether or not $L(A')$ is a code.

Step 1: If $\lambda \in L(A')$, answer **NO** and halt. Otherwise, continue.

Step 2: Construct an NFA A in restricted form with $L(A) = L(A')$.

Step 3: Construct $\hat{A}$.

Step 4: If there is a walk in $\hat{A}$ from $[q_0, q_0]$ to $[f, f]$ that passes through a semi-final state, answer **NO** and halt; otherwise, answer **YES** and halt.

To carry out Step 1, it is necessary only to determine whether there is a λ-path (i.e., a path in which all edges are labeled by λ) in A' from q_0 to some state in F. This can be accomplished in $\mathrm{O}(n)$ time using depth-first search. (See [1] for coverage of fundamental graph algorithms.)

As for Step 2, first construct A'^R (i.e., A' with its edges reversed in direction). Then apply depth-first search to A'^R (traversing only edges labeled by λ and choosing as roots of trees in the depth-first forest only nodes from F) in order to compute the set S of states in A' from which accepting states are reachable via λ-paths. To construct A, start with A', but make all its states non-accepting. Introduce a new state, f, which will be the lone accepting state in A. For every transition (p, a, s) in A', where $s \in S$ and $a \in \Sigma$, let (p, a, f) be a transition in A. Clearly, A is in restricted form, $L(A) = L(A')$, and the size of A is bounded above by cn, where c is a small constant. All this can be completed in $\mathrm{O}(n)$ time.

To carry out Step 3, construct $\langle A \times A \rangle$ and then insert the extra edges, as described in the definition of $\hat{A}$. This requires time linear in the size of $\hat{A}$, which is $\mathrm{O}(n^2)$.

As for Step 4, apply depth-first search to $\hat{A}$ (taking $[q_0, q_0]$ as the root of the depth-first tree) in order to compute the set S_1 of semi-final states reachable from $[q_0, q_0]$. Then apply depth-first search (taking $[f, f]$ as the root) to $\hat{A}^R$ (i.e., $\hat{A}$ with its edges reversed) in order to compute the set S_2 of semi-final states from which $[f, f]$ can be reached in $\hat{A}$. Determining whether to answer YES or NO reduces to determining whether or not $S_1 \cap S_2 = \emptyset$. All this can be completed in time linear in the size of $\hat{A}$, which is $\mathrm{O}(n^2)$.

References

[1] Aho, A., Hopcroft, J., and Ullman J., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] Berstel, J. and Perrin, D., *Theory of Codes*, Academic Press, Orlando, FL, 1985.

[3] Blum, E.K., Free Subsemigroups of a Free Semigroup, *Michigan Math. Journal* 12 (1965), 179-182.

[4] Gurari, E. and Ibarra, O., A Note on Finite-valued and Finitely Ambiguous Transducers, *Math. Systems Theory* 16 (1983), 61-66.

[5] Head, T. and Weber, A., Deciding Code Related Properties by means of Finite Transducers, in *Sequences II*, R.M. Capocelli, A. DeSantis, and U. Vaccaro (eds.), Springer-Verlag (1993), 260-272.

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

[7] Harrison, M.A., *Introduction to Formal Language Theory*, Addison-Wesley, Reading, MA, 1978.

[8] Lallement, G., *Semigroups and Combinatorial Theory*, Wiley, New York, 1979.

[9] Salomaa, A., *Jewels of Formal Language Theory*, Computer Science Press, Rockville, MD, 1981.

[10] Sardinas, A. and Patterson, C., A Necessary and Sufficient Condition for the Unique Decomposition of Coded Messages, *IRE Intern. Conv. Record* 8 (1953), 104-108.

[11] Spehner, J.C., Quelques Constructions et Algorithmes Relatifs aux Soux-monoides d'un Monoide Libre, *Semigroup Forum* 9 (1975), 334-353.