

THE COMPLEXITY OF A COUNTING FINITE-STATE AUTOMATON

Craig A. Rich
Computer Science Department
California State Polytechnic University, Pomona
Pomona, CA 91768-4034, USA

Giora Slutzki
Computer Science Department
Iowa State University
Ames, IA 50011, USA

Abstract. A counting finite-state automaton is a nondeterministic finite-state automaton which, on an input over its input alphabet, (magically) writes in binary the number of accepting computations on the input. We examine the complexity of computing the counting function of an NFA, and the complexity of recognizing its range as a set of binary strings. We also consider the pumping behavior of counting finite-state automata. The class of functions computed by counting NFA's

- (1) includes a class of functions computed by deterministic finite-state transducers;
- (2) is contained in the class of functions computed by polynomially time- and linearly space-bounded Turing transducers;
- (3) includes a function whose range is the composite numbers.

1. Introduction

The counting finite-state automaton—or counting NFA—is a finite-state analogue of the counting Turing Machine of Valiant [8]. It is known that the class $\#P$ of functions computed by polynomially time-bounded counting TMs includes the class FP of functions computed by polynomially time-bounded Turing transducers; however, it is not known if this inclusion is proper. Valiant [8,9] has shown several functions to be complete for $\#P$, and these functions in $\#P$ are not computable in polynomial time if $P \neq NP$. These results suggest that FP is properly included in $\#P$.

We consider finite-state analogues of these questions. We show that the class $\#NFA$ of functions computed by counting NFAs includes a class $\#DFT$ of counting functions computed by deterministic finite-state transducers. Although it is not known whether $FP \neq \#P$, we show that $\#DFT$ is properly included in $\#NFA$ by exhibiting

a counting NFA whose range as a set of binary strings is not context-free, whereas the ranges of deterministic finite-state transducers are regular [2]. While some functions in $\#P$ are apparently not computable in polynomial time, we show that functions in $\#NFA$ can be computed using time polynomial and space linear in the length of the input.

Since functions in $\#DFT$ have ranges which are regular and functions in $\#NFA$ have ranges which are not necessarily context-free, it is natural to investigate the complexity of counting NFA ranges. Intuitively, one might expect the range of a counting NFA to be efficiently recognizable simply because it is a finite-state model, but that is apparently not the case. We establish an upper bound by showing that the range of a counting NFA is recognizable nondeterministically using space linear in the length of the input, i.e., a context-sensitive language. We suggest an intractable lower bound by showing that the composite numbers—which are not known to be in P —are the range of a counting NFA.

In §2, we give notational conventions and formally define the counting function of an NFA. In §3, we show that the counting functions computed by deterministic finite-state transducers are properly included among those computed by nondeterministic finite-state automata, and give a counting NFA whose range is not context-free. In §4, we examine the complexity of computing the counting function of an NFA, and the complexity of recognizing its range as a set of binary strings. In §5, we consider the pumping behavior of a counting finite-state automaton. For a fixed input string, we show that the number of accepting computations—considered as a function of the number of times a fixed substring is pumped—satisfies a homogeneous linear recurrence equation of finite degree having integer coefficients.

2. Preliminary Definitions

In this section, we present notational conventions and our notions of counting function computed by finite-state automata. A *string* x is a finite sequence of *symbols* from a finite *alphabet*. The *length* of x , denoted $|x|$, is the number of symbols composing x . The *empty string*, denoted ϵ , is the string having length 0. The *concatenation* of two strings x and y is the string consisting of the symbols of x followed by the symbols of y , denoted xy . A *language* L is a set of strings over an alphabet, and $\|L\|$ denotes the cardinality of L . The empty set is denoted by ϕ ; the set of integers $\{\dots, -1, 0, 1, \dots\}$ is denoted by \mathcal{Z} ; and the set of natural numbers $\{0, 1, 2, \dots\}$ is denoted by \mathcal{N} .

In this work, we frequently consider natural numbers as binary strings and vice versa. Formally, these conversions are functions $s: \mathcal{N} \rightarrow \{0, 1\}^*$ and $\#: \{0, 1\}^* \rightarrow \mathcal{N}$ defined by

$$\begin{aligned} s(k) &= \text{the binary representation of } k \text{ without leading zeroes,} \\ \#(x) &= \text{the number represented in binary by } x. \end{aligned}$$

Note that $s(0) = \epsilon$. We extend s and $\#$ to sets of natural numbers and binary strings in the usual way by defining $s(K) = \{s(k) \mid k \in K\}$, and $\#(L) = \{\#(x) \mid x \in L\}$.

A *nondeterministic finite automaton (NFA)* is a 5-tuple $M = (Q, \Sigma, \delta, I, F)$, where Q is a finite set of *states*; Σ is a finite *input alphabet*; δ is a *transition function* from

$Q \times \Sigma^*$ to subsets of Q ; $I \subseteq Q$ is a set of *initial states*; and $F \subseteq Q$ is a set of *final states*. The *counting function* $\#\delta: Q \times \Sigma^* \rightarrow \mathcal{N}$ is defined recursively by

$$\begin{aligned} \#\delta(q, \epsilon) &= \begin{cases} 1, & \text{if } q \in F; \\ 0, & \text{if } q \notin F, \end{cases} \\ \#\delta(q, \sigma x) &= \sum_{p \in \delta(q, \sigma)} \#\delta(p, x). \end{aligned}$$

The *counting function* $\#M: \Sigma^* \rightarrow \mathcal{N}$ is defined by

$$\#M(x) = \sum_{q \in I} \#\delta(q, x).$$

Intuitively, the counting function $\#M(x)$ ($\#\delta(q, x)$) is the number of accepting computations of M on input x (starting from state q). We extend the counting functions to languages $L \subseteq \Sigma^*$ in the usual way by defining $\#\delta(q, L) = \{ \#\delta(q, x) \mid x \in L \}$, and $\#M(L) = \{ \#M(x) \mid x \in L \}$. We consider the range of a counting NFA M to be the set of binary strings $s(\#M(\Sigma^*))$. The class of counting functions of NFAs is defined by $\#\text{NFA} = \{ \#M \mid M \text{ is an NFA} \}$, and the class of their ranges is defined by $\text{range}(\#\text{NFA}) = \{ s(\#M(\Sigma^*)) \mid M \text{ is an NFA with input alphabet } \Sigma \}$.

We also want to consider a deterministic counterpart of the counting NFA which produces a binary string by transduction rather than counting accepting computations. Intuitively, our deterministic finite-state transducer is a special case of the deterministic Generalized Sequential Machine (GSM) [1] in which the output alphabet is fixed to be $\{0, 1\}$ and all states are considered final, so that its computation on any input produces a binary string.

Formally, a *deterministic finite-state transducer (DFT)* is a 5-tuple $D = (Q, \Sigma, \delta, \lambda, q_1)$, where Q is a finite set of *states*; Σ is a finite *input alphabet*; $\delta: Q \times \Sigma \rightarrow Q$ is a *transition function*; $\lambda: Q \times \Sigma \rightarrow \{0, 1\}^*$ is an *output function*; and $q_1 \in Q$ is the *initial state*. The transition function and output function are extended to $\delta: Q \times \Sigma^* \rightarrow Q$ and $\lambda: Q \times \Sigma^* \rightarrow \{0, 1\}^*$, defined recursively by

$$\begin{aligned} \delta(q, \epsilon) &= q, \\ \delta(q, \sigma x) &= \delta(\delta(q, \sigma), x); \\ \lambda(q, \epsilon) &= \epsilon, \\ \lambda(q, \sigma x) &= \lambda(q, \sigma)\lambda(\delta(q, \sigma), x). \end{aligned}$$

The *output function* $D: \Sigma^* \rightarrow \{0, 1\}^*$ is defined by $D(x) = \lambda(q_1, x)$ and the *counting function* $\#D: \Sigma^* \rightarrow \mathcal{N}$ is defined by $\#D(x) = \#(D(x))$. Intuitively, the counting function $\#D(x)$ is the number represented by the binary string produced by the transduction of D on input x . We extend the output and counting functions to languages $L \subseteq \Sigma^*$ in the usual way. We consider the range of a DFT to be the set of binary strings $s(\#D(\Sigma^*))$. Note that it can be obtained from $D(\Sigma^*)$ by truncating the leading zeroes of each string. The class of counting functions of DFTs is defined by $\#\text{DFT} = \{ \#D \mid D \text{ is a DFT} \}$, and the class of their ranges is defined by $\text{range}(\#\text{DFT}) = \{ s(\#D(\Sigma^*)) \mid D \text{ is a DFT with input alphabet } \Sigma \}$.

3. Inclusions among Counting Functions and their Ranges

In this section, we show that the counting functions computed by deterministic finite-state transducers are properly included among those computed by nondeterministic finite-state automata, and give a counting NFA whose range is not context-free. We denote the class of regular and context-free languages by REG and CFL, respectively. Since the range of every deterministic Generalized Sequential Machine (DGSM) is regular [2], and a DFT is a special case of a DGSM, it follows that the range of a DFT (with leading zeroes truncated) is regular.

Theorem 3.1. $\text{range}(\#DFT) \subseteq \text{REG}$.

Theorem 3.2. $\#DFT \subseteq \#NFA$.

Proof. Let $D = (Q, \Sigma, \delta, \lambda, q_1)$ be a DFT, and construct an NFA M with input alphabet Σ such that for $x \in \Sigma^*$, $\#M(x) = \#D(x)$. Suppose $Q = \{q_1, \dots, q_s\}$, and let $l = \max\{|\lambda(q, \sigma)| \mid q \in Q \wedge \sigma \in \Sigma\}$. Let $M = (Q', \Sigma, \delta', \{q_1\}, F)$, where $Q' = Q \cup \{q_i^j \mid 1 \leq i \leq s \wedge 1 \leq j \leq 2^l\}$, $F = \{q_i^j \mid 1 \leq i \leq s \wedge 1 \leq j \leq 2^l\}$, and δ' is defined by

$$\begin{aligned} \delta'(q_i, \sigma) &= \{\delta(q_i, \sigma)\} \cup \{\delta(q_i, \sigma)^j \mid 1 \leq j \leq \#(\lambda(q_i, \sigma))\}, \\ \delta'(q_i^k, \sigma) &= \{\delta(q_i, \sigma)^j \mid 1 \leq j \leq 2^{|\lambda(q_i, \sigma)|}\}. \end{aligned}$$

First we prove, by induction on $|x|$, that $\#\delta'(q_i^k, x) = 2^{|\lambda(q_i, x)|}$. For the empty string ϵ ,

$$\#\delta'(q_i^k, \epsilon) = \begin{cases} 1, & \text{if } q_i^k \in F; \\ 0, & \text{if } q_i^k \notin F \end{cases} = 1 = 2^{|\epsilon|} = 2^{|\lambda(q_i, \epsilon)|},$$

and for strings σx of length at least 1,

$$\begin{aligned} \#\delta'(q_i^k, \sigma x) &= \sum_{p \in \delta'(q_i^k, \sigma)} \#\delta'(p, x) \\ &= \sum_{1 \leq j \leq 2^{|\lambda(q_i, \sigma)|}} \#\delta'(\delta(q_i, \sigma)^j, x) \\ &= 2^{|\lambda(q_i, \sigma)|} \cdot 2^{|\lambda(\delta(q_i, \sigma), x)|} \\ &= 2^{|\lambda(q_i, \sigma)\lambda(\delta(q_i, \sigma), x)|} \\ &= 2^{|\lambda(q_i, \sigma x)|}. \end{aligned}$$

Next we prove, by induction on $|x|$, the claim (*) $\#\delta'(q_i, x) = \#(\lambda(q_i, x))$. For the empty string ϵ ,

$$\#\delta'(q_i, \epsilon) = \begin{cases} 1, & \text{if } q_i \in F; \\ 0, & \text{if } q_i \notin F \end{cases} = 0 = \#(\epsilon) = \#(\lambda(q_i, \epsilon)),$$

and for strings σx of length at least 1,

$$\begin{aligned}
\#\delta'(q_i, \sigma x) &= \sum_{p \in \delta'(q_i, \sigma)} \#\delta'(p, x) \\
&= \left(\sum_{1 \leq j \leq \#(\lambda(q_i, \sigma))} \#\delta'(\delta(q_i, \sigma)^j, x) \right) + \#\delta'(\delta(q_i, \sigma), x) \\
&= \#(\lambda(q_i, \sigma)) \cdot 2^{|\lambda(\delta(q_i, \sigma), x)|} + \#(\lambda(\delta(q_i, \sigma), x)) \\
&= \#(\lambda(q_i, \sigma)\lambda(\delta(q_i, \sigma), x)) \\
&= \#(\lambda(q_i, \sigma x)).
\end{aligned}$$

If we take $i = 1$ in (*), then

$$\#M(x) = \#\delta'(q_1, x) = \#(\lambda(q_1, x)) = \#(D(x)) = \#D(x). \quad \blacksquare$$

Lemma 3.3. $\#NFA$ is closed under addition and multiplication.

Proof idea. (addition) Let $M' = (Q', \Sigma, \delta', I', F')$, $M'' = (Q'', \Sigma, \delta'', I'', F'')$ be NFAs, and construct an NFA M with input alphabet Σ such that for $x \in \Sigma^*$, $\#M(x) = \#M'(x) + \#M''(x)$. M is obtained by a disjoint union construction. The states, transitions, initial states, and final states of M are the disjoint unions of those in M' and M'' .

(multiplication) Let $M' = (Q', \Sigma, \delta', I', F')$, $M'' = (Q'', \Sigma, \delta'', I'', F'')$ be NFAs, and construct an NFA M with input alphabet Σ such that for $x \in \Sigma^*$, $\#M(x) = \#M'(x) \cdot \#M''(x)$. M is obtained by a cartesian product construction. The states, transitions, initial states, and final states of M are the cartesian products of those in M' and M'' . \blacksquare

In this research, we give two examples of counting NFAs whose ranges are not context-free. The first is presented here, and the second—whose range is the binary encodings of the composite numbers—is presented in §4.

Example. A counting NFA whose range is $L = \{1^n 0^n 1^n \mid n \in \mathcal{N}\}$.

We construct an NFA M with input alphabet $\{0\}$ such that $s(\#M(0^*)) = L$. For $k \in \mathcal{N}$, define an NFA $M_k = (Q, \{0\}, \delta, I, F)$, where $Q = \{q_1, \dots, q_{2k}, p_1, \dots, p_{2k+1}\}$, $I = \{q_1\}$, $F = \{p_1, \dots, p_{2k+1}\}$, and δ is defined by

$$\begin{aligned}
\delta(q_i, 0) &= \{q_1, \dots, q_{2k}, p_1, \dots, p_{2k}\}, \\
\delta(p_i, 0) &= \{p_1, \dots, p_{2k+1}\}.
\end{aligned}$$

First we prove, by induction on n , that $\#\delta(p_i, 0^n) = 2^{(k+1)n}$. For $n = 0$,

$$\#\delta(p_i, \epsilon) = \begin{cases} 1, & \text{if } p_i \in F; \\ 0, & \text{if } p_i \notin F \end{cases} = 1 = 2^{(k+1)0},$$

and for $n > 0$,

$$\begin{aligned}
\#\delta(p_i, 0^n) &= \sum_{p \in \delta(p_i, 0)} \#\delta(p, 0^{n-1}) \\
&= \sum_{1 \leq j \leq 2^{k+1}} \#\delta(p_j, 0^{n-1}) \\
&= 2^{k+1} \cdot 2^{(k+1)(n-1)} = 2^{(k+1)n}.
\end{aligned}$$

Next we prove, by induction on n , the claim (*) $\#\delta(q_i, 0^n) = 2^{(k+1)n} - 2^{kn}$. For $n = 0$,

$$\#\delta(q_i, \epsilon) = \begin{cases} 1, & \text{if } q_i \in F; \\ 0, & \text{if } q_i \notin F \end{cases} = 0 = 2^{(k+1)0} - 2^{k \cdot 0},$$

and for $n > 0$,

$$\begin{aligned} \#\delta(q_i, 0^n) &= \sum_{p \in \delta(q_i, 0)} \#\delta(p, 0^{n-1}) \\ &= \left(\sum_{1 \leq j \leq 2^k} \#\delta(q_j, 0^{n-1}) \right) + \left(\sum_{1 \leq j \leq 2^k} \#\delta(p_j, 0^{n-1}) \right) \\ &= 2^k \cdot (2^{(k+1)(n-1)} - 2^{k(n-1)}) + 2^k \cdot 2^{(k+1)(n-1)} \\ &= 2^{(k+1)n} - 2^{kn}. \end{aligned}$$

Using Lemma 3.3, construct an NFA M with input alphabet $\{0\}$ such that for $n \in \mathcal{N}$, $\#M(0^n) = \#M_2(0^n) + \#M_0(0^n)$. If we take $i = 1$ in (*), then

$$\begin{aligned} s(\#M(0^*)) &= \{ s(\#M(0^n)) \mid n \in \mathcal{N} \} \\ &= \{ s(\#M_2(0^n) + \#M_0(0^n)) \mid n \in \mathcal{N} \} \\ &= \{ s((2^{3n} - 2^{2n}) + (2^n - 2^0)) \mid n \in \mathcal{N} \} \\ &= \{ s(8^n - 4^n + 2^n - 1) \mid n \in \mathcal{N} \} \\ &= \{ 1^n 0^n 1^n \mid n \in \mathcal{N} \} = L. \quad \blacksquare \end{aligned}$$

Corollary 3.4. $\text{range}(\#\text{NFA}) \not\subseteq \text{CFL}$.

Corollary 3.5. $\text{range}(\#\text{DFT}) \subset \text{range}(\#\text{NFA})$ ($\text{range}(\#\text{DFT})$ is properly contained in $\text{range}(\#\text{NFA})$).

Proof. It follows from Theorems 3.1, 3.2, Corollary 3.4, and the fact that $\text{REG} \subset \text{CFL}$. \blacksquare

4. The Complexity of Counting Functions and their Ranges

In this section, we examine the complexity of computing the counting function of an NFA, and the complexity of recognizing its range. The latter problem—deciding whether a given binary string represents the number of accepting computations on some input—is considered both for a fixed NFA and when the NFA is given as an additional parameter. We show that a fixed counting NFA's range is context-sensitive, and suggest an intractible lower bound by showing that the composite numbers—which are not known to be in P—are the range of a counting NFA. The second of these is called the range membership problem for counting NFAs and is shown to be PSPACE-complete.

An important tool which we use in solving these problems is a matrix algebraic characterization of the counting function of an NFA which allows us to compute it in polynomial time and linear space. Let $M = (Q, \Sigma, \delta, I, F)$ be an NFA with state set $Q = \{q_1, \dots, q_s\}$, and let $x = \sigma_n \dots \sigma_1 \in \Sigma^*$. For each $\sigma \in \Sigma$, let \vec{e} , A^σ , \vec{f} be the $1 \times s$, $s \times s$, $s \times 1$ matrices defined by

$$\vec{e} = (e_1 \ e_2 \ \dots \ e_s), \quad \text{where } e_j = \begin{cases} 1, & \text{if } q_j \in I; \\ 0, & \text{if } q_j \notin I, \end{cases}$$

$$A^\sigma = \begin{pmatrix} A_{11}^\sigma & A_{12}^\sigma & \dots & A_{1s}^\sigma \\ A_{21}^\sigma & A_{22}^\sigma & \dots & A_{2s}^\sigma \\ \vdots & \vdots & \ddots & \vdots \\ A_{s1}^\sigma & A_{s2}^\sigma & \dots & A_{ss}^\sigma \end{pmatrix}, \quad \text{where } A_{ij}^\sigma = \begin{cases} 1, & \text{if } q_j \in \delta(q_i, \sigma); \\ 0, & \text{if } q_j \notin \delta(q_i, \sigma), \end{cases}$$

$$\vec{f} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_s \end{pmatrix}, \quad \text{where } f_i = \#\delta(q_i, \epsilon).$$

The symbol $*$ denotes the usual matrix multiplication. The following lemma and its proof are well-known, and can also be proved using formal power series, as in Salomaa and Soittola [4].

Lemma 4.1. $\vec{e} * A^{\sigma_n} * \dots * A^{\sigma_1} * \vec{f} = \#M(x)$.

Proof. We prove, by induction on n , the following claim for $1 \leq i \leq s$:

$$(*) \quad (A^{\sigma_n} * \dots * A^{\sigma_1} * \vec{f})_i = \#\delta(q_i, \sigma_n \dots \sigma_1).$$

For $n = 0$, $(\vec{f})_i = f_i = \#\delta(q_i, \epsilon)$, and for $n > 0$,

$$\begin{aligned} (A^{\sigma_n} * \dots * A^{\sigma_1} * \vec{f})_i &= \sum_{j=1}^s A_{ij}^{\sigma_n} \cdot (A^{\sigma_{n-1}} * \dots * A^{\sigma_1} * \vec{f})_j \\ &= \sum_{j=1}^s A_{ij}^{\sigma_n} \cdot \#\delta(q_j, \sigma_{n-1} \dots \sigma_1) \\ &= \sum_{q_j \in \delta(q_i, \sigma_n)} \#\delta(q_j, \sigma_{n-1} \dots \sigma_1) \\ &= \#\delta(q_i, \sigma_n \dots \sigma_1). \end{aligned}$$

Applying $(*)$, we have

$$\begin{aligned} \vec{e} * A^{\sigma_n} * \dots * A^{\sigma_1} * \vec{f} &= \sum_{j=1}^s \vec{e}_j \cdot (A^{\sigma_n} * \dots * A^{\sigma_1} * \vec{f})_j \\ &= \sum_{j=1}^s \vec{e}_j \cdot \#\delta(q_j, x) \\ &= \sum_{q_j \in I} \#\delta(q_j, x) \\ &= \#M(x). \quad \blacksquare \end{aligned}$$

The algebraic characterization of Lemma 4.1 gives us the following algorithm for computing $\#M(x)$ which processes the symbols of x from right to left, producing an s -entry column vector after each of n matrix multiplications.

```

input  $x$ ;   $\{= \sigma_n \dots \sigma_1 \in \Sigma^*\}$ 
 $\vec{v} := \vec{f}$ ;
for  $i := 1$  to  $n$  do
     $\vec{v} := A^{\sigma_i} * \vec{v}$ ;
output  $\vec{e} * \vec{v}$ 

```

After n multiplications, we obtain $A^{\sigma_n} * \dots * A^{\sigma_1} * \vec{f}$, whose i th entry is $\#\delta(q_i, x)$. This computation can be done in time polynomial in n and, since $\#\delta(q_i, x) \leq s^n$, each entry can be represented in binary using space linear in n .

In the remainder of this section, we turn our attention to the ranges of counting NFAs. We apply the method of computing $\#M(x)$ given by the previous algorithm to show that the range $s(\#M(\Sigma^*))$ of a counting NFA M is context-sensitive, i.e., is in $\text{NSPACE}(n)$.

Given a binary string y , how can we decide if $y \in s(\#M(\Sigma^*))$? That is, how can we decide if y represents the number of accepting computations of M on some input x ? A first approach using Lemma 4.1 is to guess symbols $\sigma_1, \dots, \sigma_n$ of x from right to left, computing after each guess a column vector \vec{v} whose i th entry is $v_i = \#\delta(q_i, x)$, and accepting if and only if y is the binary representation of $\vec{e} * \vec{v} = \#M(x)$:

```

input  $y$ ;
 $\vec{v} := \vec{f}$ ;
while true do
    begin
        if  $s(\vec{e} * \vec{v}) = y$  then accept;
        guess  $\sigma \in \Sigma$ ;
         $\vec{v} := A^\sigma * \vec{v}$ 
    end

```

Some computations of this nondeterministic algorithm may not halt and will require an unbounded amount of space in which to store the entries of \vec{v} . In the following development, we show how to impose a linear space bound on the computations of this algorithm by placing a cap on the size of the entries of \vec{v} .

Let $y \in \{0, 1\}^*$. We define $\text{cap}_y: \mathcal{N} \rightarrow \mathcal{N}$ by $\text{cap}_y(m) = \min\{m, \#(y) + 1\}$. We extend cap_y to matrices of natural numbers by applying cap_y to each entry of the matrix. We will need the following properties of the cap function in order to impose a bound on the space required by the previous algorithm and maintain its correctness.

Lemma 4.2. Let $y \in \{0, 1\}^*$; $m, l \in \mathcal{N}$; and A, B compatible matrices of natural numbers.

- (1) $s(\text{cap}_y(m)) = y \iff s(m) = y$
- (2) $\text{cap}_y(m + l) = \text{cap}_y(\text{cap}_y(m) + \text{cap}_y(l))$
- (3) $\text{cap}_y(m \cdot l) = \text{cap}_y(m \cdot \text{cap}_y(l))$
- (4) $\text{cap}_y(A * B) = \text{cap}_y(A * \text{cap}_y(B))$

Proof. (1) $s(\text{cap}_y(m)) = y \iff s(\min\{m, \#(y) + 1\}) = y \iff s(m) = y$.

(2) We consider two cases. If $m + l > \#(y)$, then $\text{cap}_y(m + l) = \text{cap}_y(\text{cap}_y(m) + \text{cap}_y(l)) = \#(y) + 1$. If $m + l \leq \#(y)$, then $\text{cap}_y(m + l) = \text{cap}_y(\text{cap}_y(m) + \text{cap}_y(l)) = m + l$.

(3) Proof is similar to (2).

(4)

$$\begin{aligned}
\text{cap}_y(A * B)_{ik} &= \text{cap}_y\left(\sum_j A_{ij} \cdot B_{jk}\right) \\
&= \text{cap}_y\left(\sum_j \text{cap}_y(A_{ij} \cdot B_{jk})\right), \quad \text{by (2);} \\
&= \text{cap}_y\left(\sum_j \text{cap}_y(A_{ij} \cdot \text{cap}_y(B_{jk}))\right), \quad \text{by (3);} \\
&= \text{cap}_y\left(\sum_j A_{ij} \cdot \text{cap}_y(B_{jk})\right), \quad \text{by (2);} \\
&= \text{cap}_y(A * \text{cap}_y(B))_{ik}. \quad \blacksquare
\end{aligned}$$

Theorem 4.3. $\text{range}(\#NFA) \subseteq \text{CSL}$.

Proof. We show that for a fixed NFA M with input alphabet Σ , $s(\#M(\Sigma^*)) \in \text{NSPACE}(n)$. Consider the following modification of our previous algorithm which decides whether or not $y \in s(\#M(\Sigma^*))$:

```

input  $y$ ;
 $\vec{v} := \vec{f}$ ;
while true do
  begin
    if  $s(\text{cap}_y(\vec{e} * \vec{v})) = y$  then accept;
    guess  $\sigma \in \Sigma$ ;
     $\vec{v} := \text{cap}_y(A^\sigma * \vec{v})$ 
  end

```

The matrices \vec{e} , A^σ , and \vec{f} can be kept in finite control and the space required by \vec{v} is $O(|y|)$, since its entries are at most $\#(y) + 1$; therefore, this algorithm can be implemented by a nondeterministic linear space-bounded Turing machine. To show correctness, let $\sigma_1, \dots, \sigma_n$ be a sequence of guesses of the algorithm and $x = \sigma_n \dots \sigma_1$. By Lemma 4.2(4), the value of \vec{v} at the beginning of the while-loop after guessing x will be

$$\begin{aligned}
\vec{v} &= \text{cap}_y(A^{\sigma_n} * \text{cap}_y(A^{\sigma_{n-1}} * \dots * \text{cap}_y(A^{\sigma_1} * \vec{f}) \dots)) \\
&= \text{cap}_y(A^{\sigma_n} * \dots * A^{\sigma_1} * \vec{f}).
\end{aligned}$$

By this observation, Lemma 4.1, and Lemma 4.2(1,4), we have

$$\begin{aligned}
s(\text{cap}_y(\vec{e} * \vec{v})) = y &\iff s(\text{cap}_y(\vec{e} * \text{cap}_y(A^{\sigma_n} * \dots * A^{\sigma_1} * \vec{f}))) = y \\
&\iff s(\text{cap}_y(\vec{e} * A^{\sigma_n} * \dots * A^{\sigma_1} * \vec{f})) = y \\
&\iff s(\text{cap}_y(\#M(x))) = y \\
&\iff s(\#M(x)) = y,
\end{aligned}$$

so the algorithm accepts y if and only if $s(\#M(x)) = y$, for some $x \in \Sigma^*$. \blacksquare

It is interesting to consider whether the information in this algorithm can be further compressed into space which is logarithmic in $|y|$, giving us an $\text{NSPACE}(\log(|y|))$

algorithm for recognizing the range of a counting NFA. In the following example, we give evidence that, if possible, it will be difficult to achieve, by showing that the composite numbers—which are not known to be in P—are the range of a counting NFA.

Example. A counting NFA whose range is $\text{Composites} \cup \{0\}$.

We construct an NFA M with input alphabet Σ such that $\#M(\Sigma^*) = \text{Composites} \cup \{0\}$. First, construct an NFA M' with input alphabet $\Sigma = \{0, 1\}$ such that $\#M'(0^m 10^l) = m \cdot l$. Let M' be the NFA pictured in the transition graph of Figure 4.1.

Figure 4.1. A counting NFA which multiplies unary numbers

We prove, by induction on l , the following claims:

$$\begin{aligned}\#\delta(q_1, 0^l) &= l; \\ \#\delta(q_2, 0^l) &= 1.\end{aligned}$$

For $l = 0$,

$$\begin{aligned}\#\delta(q_1, \epsilon) &= \begin{cases} 1, & \text{if } q_1 \in F; \\ 0, & \text{if } q_1 \notin F \end{cases} = 0; \\ \#\delta(q_2, \epsilon) &= \begin{cases} 1, & \text{if } q_2 \in F; \\ 0, & \text{if } q_2 \notin F \end{cases} = 1,\end{aligned}$$

and for $l > 0$,

$$\begin{aligned}\#\delta(q_1, 0^l) &= \sum_{p \in \delta(q_1, 0)} \#\delta(p, 0^{l-1}) \\ &= \#\delta(q_1, 0^{l-1}) + \#\delta(q_2, 0^{l-1}) \\ &= (l-1) + 1 = l; \\ \#\delta(q_2, 0^l) &= \sum_{p \in \delta(q_2, 0)} \#\delta(p, 0^{l-1}) \\ &= \#\delta(q_2, 0^{l-1}) = 1.\end{aligned}$$

Next we prove, by induction on m , the following claims:

$$\begin{aligned}\#\delta(q_1, 0^m 10^l) &= m \cdot l; \\ \#\delta(q_2, 0^m 10^l) &= l.\end{aligned}$$

For $m = 0$,

$$\begin{aligned}\#\delta(q_1, 10^l) &= \sum_{p \in \delta(q_1, 1)} \#\delta(p, 0^l) = 0 = 0 \cdot l; \\ \#\delta(q_2, 10^l) &= \sum_{p \in \delta(q_2, 1)} \#\delta(p, 0^l) \\ &= \#\delta(q_1, 0^l) = l,\end{aligned}$$

and for $m > 0$,

$$\begin{aligned}\#\delta(q_1, 0^m 10^l) &= \sum_{p \in \delta(q_1, 0)} \#\delta(p, 0^{m-1} 10^l) \\ &= \#\delta(q_1, 0^{m-1} 10^l) + \#\delta(q_2, 0^{m-1} 10^l) \\ &= (m-1) \cdot l + l = m \cdot l; \\ \#\delta(q_2, 0^m 10^l) &= \sum_{p \in \delta(q_2, 0)} \#\delta(p, 0^{m-1} 10^l) \\ &= \#\delta(q_2, 0^{m-1} 10^l) = l.\end{aligned}$$

Therefore, we have $\#M'(0^m 10^l) = \#\delta(q_1, 0^m 10^l) = m \cdot l$. Consider the regular language $R = \{0^m 10^l \mid m, l \geq 2\}$. Let M'' be a DFA which accepts R . Then

$$\#M''(0^m 10^l) = \begin{cases} 1, & \text{if } m, l \geq 2; \\ 0, & \text{otherwise.} \end{cases}$$

Using Lemma 3.3, construct an NFA M such that $\#M(x) = \#M'(x) \cdot \#M''(x)$.

$$\begin{aligned}\#M(0^m 10^l) &= \#M'(0^m 10^l) \cdot \#M''(0^m 10^l) \\ &= \begin{cases} m \cdot l, & \text{if } m, l \geq 2; \\ 0, & \text{otherwise,} \end{cases}\end{aligned}$$

so $\#M(\Sigma^*) = \text{Composites} \cup \{0\}$. ■

When the range membership problem is considered as a function of both a given NFA and binary string, we are able to pinpoint its complexity by giving a completeness result for PSPACE.

Range Membership

Instance: M , an NFA with input alphabet Σ ; $y \in \{0, 1\}^*$.

Question: $y \in s(\#M(\Sigma^*))$?

Theorem 4.4. Range Membership is PSPACE-complete.

Proof. We have shown in Theorem 4.3 that membership can be decided nondeterministically using space $O(\|Q\| \cdot |y|)$, where $\|Q\|$ is the number of states in M . By Savitch's Theorem, Range Membership $\in \text{NSPACE}(n^2) \subseteq \text{PSPACE}$. We show hardness by logspace reduction from the nonuniversality problem for NFAs, which was proved PSPACE-complete by Stockmeyer and Meyer [6,7]. Let M be an NFA with input alphabet Σ .

$$\begin{aligned}
L(M) \neq \Sigma^* &\iff \exists x \in \Sigma^*, x \notin L(M) \\
&\iff \exists x \in \Sigma^*, \#M(x) = 0 \\
&\iff 0 \in \#M(\Sigma^*) \\
&\iff \epsilon \in s(\#M(\Sigma^*)). \blacksquare
\end{aligned}$$

5. Pumping Behavior and Linear Recurrences

In this section, we consider the pumping behavior of a counting finite-state automaton. For a fixed input string, we show that the number of accepting computations—considered as a function of the number of times a fixed substring of the input is pumped—satisfies a homogeneous linear recurrence equation of finite degree having integer coefficients. We precede this result with some relevant definitions and facts from the theories of recurrence equations and matrices.

Let $g: \mathcal{N} \rightarrow \mathcal{N}$. g satisfies a homogeneous linear recurrence equation of degree s having integer coefficients if there exist $a_1, \dots, a_s \in \mathcal{Z}$ such that for $n \in \mathcal{N}$,

$$g(n + s) = \sum_{k=1}^s a_k \cdot g(n + s - k).$$

Let A be an $s \times s$ matrix of integers, and I be the $s \times s$ identity matrix with 1's on the diagonal and 0's elsewhere. The *characteristic polynomial* of A is the polynomial p defined by $p(\lambda) = \det(A - \lambda \cdot I)$, where \det is the determinant function. Note that the characteristic polynomial is of degree s and has integer coefficients, since A has integer entries. The *characteristic equation* of A is the equation $\det(A - \lambda \cdot I) = 0$. The characteristic polynomial is said to be *monic*, since the coefficient of λ^s is $(-1)^s = \pm 1$; therefore, the characteristic equation can be written as

$$\lambda^s = \sum_{k=1}^s a_k \cdot \lambda^{s-k},$$

where $a_1, \dots, a_s \in \mathcal{Z}$. One of the most important results in matrix theory is the Cayley-Hamilton Theorem, which states that a matrix satisfies its own characteristic equation. We use it to analyze the pumping behavior of a counting finite-state automaton.

Theorem 5.1. Let $M = (Q, \Sigma, \delta, I, F)$ be an NFA with state set $Q = \{q_1, \dots, q_s\}$, and let $w, x, z \in \Sigma^*$. There exist $a_1, \dots, a_s \in \mathcal{Z}$ such that for every $n \in \mathcal{N}$,

$$\#M(wx^{n+s}z) = \sum_{k=1}^s a_k \cdot \#M(wx^{n+s-k}z).$$

Proof. Let \vec{e} , A^σ , \vec{f} be defined as in section 4, and let $A^x = A^{\sigma_1} * \dots * A^{\sigma_{|x|}}$, where $x = \sigma_1 \dots \sigma_{|x|}$. We define A^w and A^z similarly. As discussed before, the characteristic equation of A^x can be written as

$$\lambda^s = \sum_{k=1}^s a_k \cdot \lambda^{s-k},$$

where $a_1, \dots, a_s \in \mathcal{Z}$. By the Cayley-Hamilton Theorem,

$$A^{x^s} = \sum_{k=1}^s a_k \cdot A^{x^{s-k}}.$$

By this observation and Lemma 4.1, we have for $1 \leq i \leq s$ and $n \in \mathcal{N}$,

$$\begin{aligned} \#\delta(q_i, wx^{n+s}z) &= (A^w * A^{x^{n+s}} * A^z * \vec{f})_i \\ &= (A^w * A^{x^n} * A^{x^s} * A^z * \vec{f})_i \\ &= \left(A^w * A^{x^n} * \left(\sum_{k=1}^s a_k \cdot A^{x^{s-k}} \right) * A^z * \vec{f} \right)_i \\ &= \left(\sum_{k=1}^s a_k \cdot \left(A^w * A^{x^n} * A^{x^{s-k}} * A^z * \vec{f} \right) \right)_i \\ &= \sum_{k=1}^s a_k \cdot (A^w * A^{x^{n+s-k}} * A^z * \vec{f})_i \\ &= \sum_{k=1}^s a_k \cdot \#\delta(q_i, wx^{n+s-k}z). \end{aligned}$$

$$\begin{aligned} \#M(wx^{n+s}z) &= \sum_{q_i \in I} \#\delta(q_i, wx^{n+s}z) \\ &= \sum_{q_i \in I} \left(\sum_{k=1}^s a_k \cdot \#\delta(q_i, wx^{n+s-k}z) \right) \\ &= \sum_{k=1}^s a_k \cdot \left(\sum_{q_i \in I} \#\delta(q_i, wx^{n+s-k}z) \right) \\ &= \sum_{k=1}^s a_k \cdot \#M(wx^{n+s-k}z). \quad \blacksquare \end{aligned}$$

Stearns and Hunt [5] showed that the number of accepting computations over all inputs of a given length—considered as a function of the length—satisfies a homogeneous linear recurrence equation of finite degree having rational coefficients. The technique of Theorem 5.1 can be used to strengthen and simplify the proof of their result, obtaining integer coefficients and a recurrence equation which is satisfied regardless of which state is considered to be the start state, by applying the Cayley-Hamilton Theorem with the matrix $A = \sum_{\sigma \in \Sigma} A^\sigma$.

6. Summary and Open Questions

We summarize some of the results contained heretofore, and ask open questions about improvements and extensions of our results.

In §3, we showed that $\text{range}(\#NFA)$ includes $\text{range}(\#DFT)$ —the ranges of deterministic finite-state transducers. It follows from the Generalized Sequential Machine results of Ginsburg and Greibach [2] that the latter is the class of all regular languages comprised of binary strings without leading zeroes. Is the class of all context-free languages comprised of binary strings without leading zeroes included in $\text{range}(\#NFA)$?

In §4, we showed that the range $s(\#M(\Sigma^*))$ of a counting NFA is in $\text{NSPACE}(n)$. How tight is this upper bound? Respecting the fact that the composite numbers are the range of a counting NFA, is there a subclass of $\text{NSPACE}(n)$ which contains $\text{range}(\#NFA)$? Are there ranges of counting NFAs which are complete for $\text{NSPACE}(n)$? NP? some other time- or space-bounded complexity class?

In §5, we showed that for a fixed input string, the number of accepting computations—considered as a function of the number of times a fixed substring of the input is pumped—satisfies a homogeneous linear recurrence equation of finite degree having integer coefficients. Does this lead to a simple pumping lemma which can be used to show that a function is not in $\#NFA$ or a language is not in $\text{range}(\#NFA)$? Which arbitrary functions satisfying linear recurrences as in Theorem 5.1 are computed by counting NFAs? That is, can we precisely characterize $\#NFA$ as a class of functions satisfying a restricted class of recurrence equations?

References

- [1] Ginsburg, S. “Examples of abstract machines,” *IEEE Trans. on Electronic Computers* **11**: 2 (1962), 132–135.
- [2] Ginsburg, S., and Greibach, S.A. “Abstract families of languages,” *Studies in Abstract Families of Languages*, pp. 1–32, Memoir No. 87, American Mathematical Society, Providence, R.I., 1969.
- [3] Hopcroft, J.E., and Ullman, J.D. “Introduction to automata theory, languages, and computation,” Addison-Wesley, Reading, Mass., 1979.
- [4] Salomaa, A., and Soittola, M. “Automata-Theoretic Aspects of Formal Power Series,” Springer-Verlag, Berlin, 1978.
- [5] Stearns, R.E., and Hunt, H.B. III. “On the equivalence and containment problems for unambiguous regular expressions, regular grammars, and finite automata,” *SIAM J. Comput.* **14** (1985), 598–611.
- [6] Stockmeyer, L.J., and Meyer, A.R. “Word problems requiring exponential time,” *Proc. Fifth Annual ACM Symposium on the Theory of Computing* (1973), 1–9.
- [7] Stockmeyer, L.J. “The Complexity of Decision Problems in Automata Theory and Logic,” Doctoral Thesis, Dept. of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, Mass., 1974.

- [8] Valiant, L.G. “The complexity of computing the permanent,” *Theor. Comput. Sci.* **8** (1979), 189–201.
- [9] Valiant, L.G. “The complexity of enumeration and reliability problems,” *SIAM J. Comput.* **8**: 3 (1979), 410–421.