

PART I. OPTIMIZING FINITE-STATE TRANSDUCERS

1. Introduction

An optimizing finite-state transducer (OFT) is a nondeterministic finite-state transducer with final states, in which states are either maximizing or minimizing. In a maximizing (minimizing) state, the optimal output is the maximum (minimum)—over all transitions on the current input symbol—of the transition output concatenated with the optimal output of the resulting state. The input is consumed from left to right, one symbol per transition.

The optimizing finite-state transducer represents a marriage of existing finite-state automata generalizations. The notion of finite-state automata as transducers first appeared in Ginsburg [5], where they are called Generalized Sequential Machines (GSMs). Chandra, Kozen, and Stockmeyer [1] introduced alternation in finite-state automata as acceptors, wherein states are either existential or universal. In an existential (universal) state, an alternating finite automaton accepts if some (every) transition results in a state in which the remaining input is accepted.

We combine the notions of transduction and alternation to obtain the OFT; however, the idea is not entirely new. The OFT model is a finite-state version of the metric Turing machine of Krentel [14], which adds optimization to the states of a polynomially time-bounded nondeterministic Turing transducer. The optimal output function that we define was motivated by his OptP functions. There is also an obvious analogy between OFTs and alternating finite automata. The maximizing (minimizing) states in OFTs correspond to existential (universal) states in the alternating finite automata. An OFT which computes the characteristic function of the language accepted by an

alternating finite automaton can be constructed in a straightforward way, exploiting the fact that “max (min)” and “or (and)” agree over the domain $\{0, 1\}$.

The introduction of transduction or alternation in finite-state automata fails to add any additional power in the following sense: Ginsburg and Greibach [6] showed that the regular languages are closed under both deterministic and nondeterministic GSM mappings, in other words, the ranges of GSM mappings are regular; and Chandra, Kozen, and Stockmeyer [1] showed that alternating finite automata accept exactly the regular languages. These results suggest the following question about optimizing finite-state transducers—is the range of an OFT necessarily regular, and if not, what is an upper bound on its complexity? We answer this by showing that the range is not necessarily regular or context-free, but can be recognized by a nondeterministic logspace Turing machine.

As Chandra, Kozen, and Stockmeyer did for alternating Turing machines, we consider OFTs whose computations have a bounded number of alternations between maximizing and minimizing states, and show that their ranges form a hierarchy in NL based on the number of alternations.

In §2, we give notational conventions and formally define the OFT. In §3, we algebraically characterize the optimal output function, providing a polynomially time-bounded algorithm for its computation. In §4, we show that the class of ranges of OFTs, which we denote $\text{range}(\text{OFT})$, is in NL and contains noncontext-free languages. In §5, we present operations under which $\text{range}(\text{OFT})$ is closed: union, concatenation, Kleene closure, and monotonic homomorphism. In §6, we present a hierarchy in NL based on the ranges of OFTs whose computations have a bounded number of alternations between maximizing and minimizing states. In §7, we consider the complexity of decision problems involving OFTs. Among these are problems which are NL-complete, NP-complete, PSPACE-complete, and RE-complete. In particular, the inequivalence

problem—whether or not two OFTs compute different optimal output functions, and the range inequivalence problem are shown to be RE-complete. The latter leads to a proof that $\text{range}(\text{OFT})$ is not effectively closed under complement, hence differs from NL, which has recently been shown to be effectively closed under complement by Immerman [11]. Finally, in §8, we summarize and present open questions about optimizing finite-state transducers.

2. Preliminary Definitions

In this section, we present notational conventions and the model of computation, the optimizing finite-state transducer. A *string* x is a finite sequence of *symbols* from an ordered, 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 . The set of strings over an alphabet is ordered *lexicographically*—shorter strings precede longer ones, and strings of equal length are ordered alphabetically. We write $x \leq y$ to denote that x is lexicographically less than or equal to y . 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 natural numbers $\{0, 1, 2, \dots\}$ is denoted by \mathcal{N} ; and the set of positive integers $\{1, 2, 3, \dots\}$ is denoted by \mathcal{Z}^+ . The following operations on languages are defined:

union	$L_1 \cup L_2 = \{x \mid x \in L_1 \vee x \in L_2\}$
intersection	$L_1 \cap L_2 = \{x \mid x \in L_1 \wedge x \in L_2\}$
concatenation	$L_1 L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$
iteration	$L^0 = \{\epsilon\}, L^{i+1} = LL^i$
Kleene closure	$L^* = \bigcup_{i \geq 0} L^i$
difference	$L_1 - L_2 = \{x \mid x \in L_1 \wedge x \notin L_2\}$
quotient	$L/y = \{x \mid xy \in L\}$
symmetric difference	$L_1 \triangle L_2 = (L_1 - L_2) \cup (L_2 - L_1)$
maximum	$\max L = \{x \mid x \in L \wedge \forall y \in L, y \leq x\}$
minimum	$\min L = \{x \mid x \in L \wedge \forall y \in L, x \leq y\}$
length	$ L = \{ x \mid x \in L\}$

Note that $\|\max L\| \leq 1$, $\|\min L\| \leq 1$, $\max L$ is empty whenever L is empty or infinite, and $\min L$ is empty whenever L is empty. Since singleton languages arise frequently in this work (e.g., as the optimal output of optimizing finite-state transducers), we suppress braces and write x for $\{x\}$ when no ambiguity results.

An *optimizing finite-state transducer (OFT)* is a 7-tuple

$$M = (Q, Q_{\max}, \Sigma, \Delta, \delta, q_1, F),$$

where Q is a finite set of *states*; $Q_{\max} \subseteq Q$ is a set of *maximizing* states ($Q_{\min} = Q - Q_{\max}$ is the set of *minimizing* states); Σ is an ordered, finite *input alphabet*; Δ is an ordered, finite *output alphabet*; δ is a *transition function* from $Q \times \Sigma$ to finite subsets of $Q \times \Delta^*$; $q_1 \in Q$ is the *initial state*; and $F \subseteq Q$ is a set of *final* states.

Hereafter, the following notational conventions are used: $p, q \in Q$; $\sigma \in \Sigma$; $a, b, c \in \Delta$; $x \in \Sigma^*$; and $w, y \in \Delta^*$. We write $\delta(q, \sigma, p)$ for $\{w \mid (p, w) \in \delta(q, \sigma)\}$, the set

of outputs of transitions from q to p on input σ . The (ab)use of δ as different functions of two and three arguments is a notational convenience. In some OFT constructions, we give $\delta(q, \sigma, p)$, and observe here that $\delta(q, \sigma)$ can be derived by $\delta(q, \sigma) = \bigcup_{p \in Q} (\{p\} \times \delta(q, \sigma, p))$. The *optimal output function* $\tilde{\delta}$ from $Q \times \Sigma^*$ to finite subsets of Δ^* is defined recursively by

$$\tilde{\delta}(q, \epsilon) = \begin{cases} \{\epsilon\}, & \text{if } q \in F; \\ \phi, & \text{if } q \notin F, \end{cases}$$

$$\tilde{\delta}(q, \sigma x) = \begin{cases} \max\left(\bigcup_{p \in Q} \delta(q, \sigma, p) \tilde{\delta}(p, x)\right), & \text{if } q \in Q_{\max}; \\ \min\left(\bigcup_{p \in Q} \delta(q, \sigma, p) \tilde{\delta}(p, x)\right), & \text{if } q \in Q_{\min}. \end{cases}$$

The *optimal output function* \tilde{M} from Σ^* to finite subsets of Δ^* is defined by $\tilde{M}(x) = \tilde{\delta}(q_1, x)$. We extend \tilde{M} to languages $L \subseteq \Sigma^*$ by defining $\tilde{M}(L) = \bigcup_{x \in L} \tilde{M}(x)$. A simple induction on $|x|$ shows that $\|\tilde{\delta}(q, x)\| \leq 1$. We also assume $\|\delta(q, \sigma, p)\| \leq 1$, since we can take, without loss of generality, \max (\min) $\delta(q, \sigma, p)$ instead of $\delta(q, \sigma, p)$ when q is maximizing (minimizing). The functions \tilde{M} and their ranges $\tilde{M}(\Sigma^*)$ are our main objects of study. We refer to the class of ranges $\tilde{M}(\Sigma^*)$, where M is an OFT with input alphabet Σ , as $\text{range}(\text{OFT})$.

An OFT can be graphically represented by a *transition diagram* in which maximizing states are drawn as triangles pointing upward, minimizing states are drawn as triangles pointing downward, the initial state is indicated by an incoming arrow without origin, the final states are drawn as double triangles, and the transition $\delta(q, \sigma, p) = w$ is denoted by an arrow from state q to state p labeled with σ/w . Occasionally, we represent states as circles to indicate that they could be maximizing or minimizing,

Figure 2.1. A transition diagram

depending on a choice in the enclosing proof. For example, consider the transition diagram in Figure 2.1.

This transition diagram represents the OFT $M = (Q, Q_{\max}, \Sigma, \Delta, \delta, q_1, F)$, where $Q = \{q_1, q_2\}$, $Q_{\max} = \{q_1\}$, $\Sigma = \{0\}$, $\Delta = \{a, b, c\} (a < b < c)$, $\delta(q_1, 0) = \{(q_1, b), (q_2, b)\}$, $\delta(q_2, 0) = \{(q_1, a), (q_2, c)\}$, q_1 is the initial state, and $F = \{q_1, q_2\}$.

A *computation* of an OFT $M = (Q, Q_{\max}, \Sigma, \Delta, \delta, q_1, F)$ on input $x = \sigma_1 \dots \sigma_n \in \Sigma^*$ is a sequence

$$p_0, w_1, p_1, w_2, p_2, \dots, w_k, p_k,$$

where (1) p_0 is the initial state q_1 ; (2) $w_i \in \delta(p_{i-1}, \sigma_i, p_i)$, for $1 \leq i \leq k$; and (3) $k = n \vee (k < n \wedge \delta(p_k, \sigma_{k+1}) = \phi)$. A computation is *accepting* if $k = n \wedge p_k \in F$. The number of *alternations* of a computation is the number of alternating sequences of maximizing and minimizing states in the computation. For example, a computation consisting of only maximizing or minimizing states has 1 alternation. An OFT is a \max_j -(\min_j -)OFT if its initial state is maximizing (minimizing), and every computation has at most j alternations. The OFT of Figure 2.1 is not a \max_j -OFT for any $j \geq 1$, since it has computations having arbitrarily many alternations. The computations of

an OFT on a particular input are perhaps best illustrated as branches of a *computation tree* having the initial state as root. For example, the computation tree of the OFT in Figure 2.1 on input $x = 00$ is given in Figure 2.2.

Figure 2.2. A computation tree

The optimal output $\widetilde{M}(x)$ can be determined by evaluating the tree “from the leaves up.” Leaves which are final states have optimal output ϵ , and those which are nonfinal have optimal output ϕ . Interior states are evaluated by taking the maximum or minimum—over all outgoing transitions—of the transition output concatenated with the optimal output of the resulting state. The optimal output of every state in the computation tree has been labeled in Figure 2.2.

3. An Algebraic Characterization

The method of computing $\widetilde{M}(x)$ by evaluating the computation tree can require time exponential in $|x|$, so we present an algebraic characterization which provides a polynomially time-bounded algorithm for computing $\widetilde{M}(x)$.

Let A, B be $m \times n, n \times p$ matrices whose entries A_{ij}, B_{jk} are finite languages. The *signature* of A is a function, $\text{sign}: \{1, \dots, m\} \rightarrow \{\max, \min\}$, which assigns max or min to each row of A . The *optimizing product* of A and B , denoted $A * B$, is the $m \times p$ matrix defined by

$$A * B = \begin{pmatrix} L_{11} & L_{12} & \dots & L_{1p} \\ L_{21} & L_{22} & \dots & L_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ L_{m1} & L_{m2} & \dots & L_{mp} \end{pmatrix}, \quad \text{where } L_{ik} = \text{sign}(i) \left(\bigcup_{j=1}^n A_{ij} B_{jk} \right).$$

$A * B$ inherits the signature of A . First, we show that optimizing product is not an associative operation. Assume the usual ordering $a < b < c$ on the underlying alphabet, and let A, B be the following $2 \times 2, 2 \times 1$ matrices having the indicated row signatures:

$$A = \begin{matrix} \max \\ \min \end{matrix} \begin{pmatrix} b & b \\ a & c \end{pmatrix}, \quad B = \begin{matrix} \max \\ \min \end{matrix} \begin{pmatrix} \epsilon \\ \epsilon \end{pmatrix}.$$

Consider $(A * A) * B$ and $A * (A * B)$:

$$\begin{aligned} (A * A) * B &= \begin{pmatrix} bb & bc \\ ab & ab \end{pmatrix} * \begin{pmatrix} \epsilon \\ \epsilon \end{pmatrix} = \begin{pmatrix} bc \\ ab \end{pmatrix}, \\ A * (A * B) &= \begin{pmatrix} b & b \\ a & c \end{pmatrix} * \begin{pmatrix} b \\ a \end{pmatrix} = \begin{pmatrix} bb \\ ab \end{pmatrix}. \end{aligned}$$

Optimizing product is not associative, since $(A * A) * B \neq A * (A * B)$. We note here without proof that if the signatures involved are exclusively max or min, then the optimizing product is associative. In general, the signatures of operand matrices

need not be the same, but in our applications, matrices will have the same signature determined by the minimizing and maximizing states of an OFT. Hereafter, we stipulate that optimizing product associates to the right. For example, $A * A * B = A * (A * B)$.

Next, we show how optimizing product can be used to compute $\widetilde{M}(x)$. Let $M = (Q, Q_{\max}, \Sigma, \Delta, \delta, q_1, F)$ be an OFT 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

$$\vec{e} = (e_1 \quad e_2 \quad \dots \quad e_s), \quad \text{where } e_j = \begin{cases} \{\epsilon\}, & \text{if } j = 1; \\ \phi, & \text{if } 1 < j \leq s, \end{cases}$$

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

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

The signatures of A^σ and \vec{f} are defined by

$$\text{sign}(i) = \begin{cases} \max, & \text{if } q_i \in Q_{\max}; \\ \min, & \text{if } q_i \in Q_{\min}, \end{cases}$$

and the signature of \vec{e} , $\text{sign}_{\vec{e}}$, is irrelevant and can be either max or min.

Theorem 3.1. $(A^{\sigma_n} * A^{\sigma_{n-1}} * \dots * A^{\sigma_1} * \vec{f})_i = \widetilde{\delta}(q_i, \sigma_n \sigma_{n-1} \dots \sigma_1)$.

Proof (by induction on n). If $n = 0$, then $(\vec{f})_i = f_i = \widetilde{\delta}(q_i, \epsilon)$, by definition. If $n > 0$, then

$$\begin{aligned} (A^{\sigma_n} * A^{\sigma_{n-1}} * \dots * A^{\sigma_1} * \vec{f})_i &= \text{sign}(i) \left(\bigcup_{j=1}^s A_{ij}^{\sigma_n} (A^{\sigma_{n-1}} * \dots * A^{\sigma_1} * \vec{f})_j \right) \\ &= \text{sign}(i) \left(\bigcup_{j=1}^s \delta(q_i, \sigma_n, q_j) \widetilde{\delta}(q_j, \sigma_{n-1} \dots \sigma_1) \right) \\ &= \widetilde{\delta}(q_i, \sigma_n \sigma_{n-1} \dots \sigma_1). \quad \blacksquare \end{aligned}$$

Corollary 3.2. $\vec{e} * A^{\sigma_n} * \dots * A^{\sigma_1} * \vec{f} = \widetilde{M}(\sigma_n \dots \sigma_1)$.

Proof.

$$\begin{aligned}
\vec{e} * A^{\sigma_n} * \dots * A^{\sigma_1} * \vec{f} &= \text{sign}_{\vec{e}} \left(\bigcup_{j=1}^s e_j(A^{\sigma_n} * \dots * A^{\sigma_1} * \vec{f})_j \right) \\
&= \text{sign}_{\vec{e}}(e_1(A^{\sigma_n} * \dots * A^{\sigma_1} * \vec{f})_1) \\
&= \text{sign}_{\vec{e}}(\widetilde{\delta}(q_1, \sigma_n \dots \sigma_1)) \\
&= \widetilde{\delta}(q_1, \sigma_n \dots \sigma_1) \\
&= \widetilde{M}(\sigma_n \dots \sigma_1). \quad \blacksquare
\end{aligned}$$

The algebraic characterization of Corollary 3.2 gives us an algorithm for computing $\widetilde{M}(x)$ which processes the symbols of x from right to left, producing an s -entry column vector after each of n optimal products. Since $|\widetilde{\delta}(q_i, \sigma_n \dots \sigma_1)|$ is $O(n)$ and each optimal product can be computed in time $O(n)$, the algorithm requires space $O(n)$ in which to store the vector and requires time $O(n^2)$.

4. The Class of Ranges

In this section, we apply the method of computing $\widetilde{\delta}(q, x)$ given by Theorem 3.1 to show that the range $\widetilde{M}(\Sigma^*)$ of an OFT M is in NL, hence, $\text{range}(\text{OFT}) \subseteq \text{NL}$. We will see in a subsequent section that this inclusion is proper.

We assume that the reader is familiar with the basic concepts of language and decidability theory, the Turing machine model, and the time- and space-bounded complexity classes $\text{DTIME}(S(n))$, $\text{NTIME}(S(n))$, $\text{DSpace}(S(n))$, and $\text{NSpace}(S(n))$. We will be concerned mainly with the following complexity classes:

$$L = \text{DSPACE}(\log n),$$

$$\text{NL} = \text{NSPACE}(\log n),$$

$$P = \bigcup_{i \geq 1} \text{DTIME}(n^i),$$

$$\text{NP} = \bigcup_{i \geq 1} \text{NTIME}(n^i),$$

$$\text{CSL} = \text{NSPACE}(n),$$

$$\text{PSPACE} = \bigcup_{i \geq 1} \text{DSPACE}(n^i) = \bigcup_{i \geq 1} \text{NSPACE}(n^i).$$

For formal definitions of these notions, refer to [8]. Recall that $L \subseteq \text{NL} \subseteq P \subseteq \text{NP} \subseteq \text{PSPACE}$ and $\text{NL} \subseteq \text{CSL} \subseteq \text{PSPACE}$.

How can we decide if $y \in \widetilde{M}(\Sigma^*)$? That is, how can we decide if there exists an $x \in \Sigma^*$ such that $\widetilde{M}(x) = y$? A first approach using Theorem 3.1 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 first entry is $v_1 = \widetilde{\delta}(q_1, \sigma_n \dots \sigma_1) = \widetilde{M}(x)$:

```

input  $y$ ;
 $\vec{v} := \vec{f}$ ;
while true do
  begin
    if  $v_1 = y$  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 space bound on the computations of this algorithm by

replacing the entries of \vec{v} by representations with respect to the input y which require space $O(\log |y|)$.

Let $w, y = a_{|y|} \dots a_1$ be strings over some alphabet. The *representation* of w with respect to y , $\text{rep}_y(w)$, has the form lr , where $l \in \mathcal{N}$ and $r \in \{\text{gt}, \text{eq}, \text{lt}\}$, and is defined by

$$\text{rep}_y(w) = \begin{cases} (|y| + 1)\text{gt}, & \text{if } |w| > |y|; \\ |w|\text{gt}, & \text{if } |w| \leq |y| \wedge (w > a_{|w|} \dots a_1); \\ |w|\text{eq}, & \text{if } |w| \leq |y| \wedge (w = a_{|w|} \dots a_1); \\ |w|\text{lt}, & \text{if } |w| \leq |y| \wedge (w < a_{|w|} \dots a_1). \end{cases}$$

In most cases, a representation $\text{rep}_y(w) = lr$ contains the length l of w and the relation r between w and a suffix of y of length $|w|$. Such a representation can be stored in binary using space $O(\log |y|)$. In practice, we may have only a representation lr of a string w' , and we would like to obtain a representation of ww' , so we extend the definition as follows:

$$\text{rep}_y(w, lr) = \begin{cases} (|y| + 1)\text{gt}, & \text{if } |w| + l > |y|; \\ (|w| + l)\text{gt}, & \text{if } |w| + l \leq |y| \wedge (w > a_{|w|+l} \dots a_{1+l}); \\ (|w| + l)r, & \text{if } |w| + l \leq |y| \wedge (w = a_{|w|+l} \dots a_{1+l}); \\ (|w| + l)\text{lt}, & \text{if } |w| + l \leq |y| \wedge (w < a_{|w|+l} \dots a_{1+l}). \end{cases}$$

Note that $\text{rep}_y(\epsilon) = 0\text{eq}$ and $\text{rep}_y(w) = \text{rep}_y(w, 0\text{eq})$. We extend rep_y to representations of languages L by defining

$$\begin{aligned} \text{rep}_y(L) &= \{ \text{rep}_y(w) \mid w \in L \}, \\ \text{rep}_y(L, R) &= \{ \text{rep}_y(w, lr) \mid w \in L \wedge lr \in R \}. \end{aligned}$$

We fix an ordering of relations $\text{lt} < \text{eq} < \text{gt}$ and define an ordering of representations by $l_1 r_1 < l_2 r_2$ iff $l_1 < l_2 \vee (l_1 = l_2 \wedge r_1 < r_2)$. Using this ordering, the operations \max and \min apply to sets of representations just as they do to languages. Since our idea is to replace strings by their representations with respect to a string y , we will need the following properties of representations.

Lemma 4.1. Let $w, y = a_{|y|} \dots a_1$ be strings over some alphabet.

- (1) $w = y \iff \text{rep}_y(w) = |y|\text{eq}$.
- (2) $w_1 \leq w_2 \implies \text{rep}_y(w_1) \leq \text{rep}_y(w_2)$.
- (3) $\text{rep}_y(w_1 w_2) = \text{rep}_y(w_1, \text{rep}_y(w_2))$.

Proof. (1) $w = y \iff |w| = |y| \wedge (w = a_{|y|} \dots a_1) \iff \text{rep}_y(w) = |y|\text{eq}$.

(2) Let $l_1 r_1 = \text{rep}_y(w_1)$, $l_2 r_2 = \text{rep}_y(w_2)$, and prove the contrapositive:

$$\begin{aligned} \text{rep}_y(w_1) > \text{rep}_y(w_2) &\iff l_1 r_1 > l_2 r_2 \\ &\iff l_1 > l_2 \vee (l_1 = l_2 \wedge r_1 > r_2) \\ &\implies |w_1| > |w_2| \vee (|w_1| = |w_2| \wedge w_1 > w_2) \\ &\iff w_1 > w_2. \end{aligned}$$

(3) We consider two cases. If $|w_1 w_2| > |y|$, then

$$\text{rep}_y(w_1 w_2) = (|y| + 1)\text{gt} = \text{rep}_y(w_1, \text{rep}_y(w_2)).$$

If $|w_1 w_2| \leq |y|$, then

$$\begin{aligned} \text{rep}_y(w_1 w_2) &= \begin{cases} |w_1 w_2|\text{gt}, & \text{if } w_1 w_2 > a_{|w_1 w_2|} \dots a_1; \\ |w_1 w_2|\text{eq}, & \text{if } w_1 w_2 = a_{|w_1 w_2|} \dots a_1; \\ |w_1 w_2|\text{lt}, & \text{if } w_1 w_2 < a_{|w_1 w_2|} \dots a_1 \end{cases} \\ &= \begin{cases} |w_1 w_2|\text{gt}, & \text{if } w_1 > a_{|w_1 w_2|} \dots a_{1+|w_2|} \vee \\ & (w_1 = a_{|w_1 w_2|} \dots a_{1+|w_2|} \wedge w_2 > a_{|w_2|} \dots a_1); \\ |w_1 w_2|\text{eq}, & \text{if } (w_1 = a_{|w_1 w_2|} \dots a_{1+|w_2|} \wedge w_2 = a_{|w_2|} \dots a_1); \\ |w_1 w_2|\text{lt}, & \text{if } w_1 < a_{|w_1 w_2|} \dots a_{1+|w_2|} \vee \\ & (w_1 = a_{|w_1 w_2|} \dots a_{1+|w_2|} \wedge w_2 < a_{|w_2|} \dots a_1) \end{cases} \\ &= \text{rep}_y(w_1, \text{rep}_y(w_2)). \quad \blacksquare \end{aligned}$$

Next, we modify the optimal product to operate on matrices whose entries are representations of finite languages instead of finite languages. Let A be an $m \times n$ matrix having signature sign whose entries A_{ij} are finite languages, B be an $n \times p$ matrix whose

entries B_{jk} are representations of finite languages, and y be a string. The *optimizing product* of A and B with respect to y , denoted $A *^y B$, is the $m \times p$ matrix defined by

$$A *^y B = \begin{pmatrix} L_{11} & L_{12} & \cdots & L_{1p} \\ L_{21} & L_{22} & \cdots & L_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ L_{m1} & L_{m2} & \cdots & L_{mp} \end{pmatrix}, \quad \text{where } L_{ik} = \text{sign}(i) \left(\bigcup_{j=1}^n \text{rep}_y(A_{ij}, B_{jk}) \right).$$

$A *^y B$ inherits the signature of A , and we stipulate that $*^y$ associates to the right.

Next, we show how $*^y$ can be used to compute $\text{rep}_y(\widetilde{M}(x))$. Let M be an OFT having state set $Q = \{q_1, \dots, q_s\}$, and let $x = \sigma_n \dots \sigma_1 \in \Sigma^*$. Let A^σ , \vec{f} be defined as in section 3, and define \vec{f}^y to be the $s \times 1$ column vector

$$\vec{f}^y = \begin{pmatrix} f_1^y \\ f_2^y \\ \vdots \\ f_s^y \end{pmatrix}, \quad \text{where } f_i^y = \text{rep}_y(f_i).$$

Lemma 4.2. $(A^{\sigma_n} *^y A^{\sigma_{n-1}} *^y \dots *^y A^{\sigma_1} *^y \vec{f}^y)_i = \text{rep}_y(\widetilde{\delta}(q_i, \sigma_n \sigma_{n-1} \dots \sigma_1))$.

Proof (by induction on n). If $n = 0$, then $(\vec{f}^y)_i = f_i^y = \text{rep}_y(f_i) = \text{rep}_y(\widetilde{\delta}(q_i, \epsilon))$, by definition. If $n > 0$, then

$$\begin{aligned} & (A^{\sigma_n} *^y A^{\sigma_{n-1}} *^y \dots *^y A^{\sigma_1} *^y \vec{f}^y)_i \\ &= \text{sign}(i) \left(\bigcup_{j=1}^s \text{rep}_y(A_{ij}^{\sigma_n}, (A^{\sigma_{n-1}} *^y \dots *^y A^{\sigma_1} *^y \vec{f}^y)_j) \right) \\ &= \text{sign}(i) \left(\bigcup_{j=1}^s \text{rep}_y(\delta(q_i, \sigma_n, q_j), \text{rep}_y(\widetilde{\delta}(q_j, \sigma_{n-1} \dots \sigma_1))) \right) \\ &= \text{sign}(i) \left(\bigcup_{j=1}^s \text{rep}_y(\delta(q_i, \sigma_n, q_j) \widetilde{\delta}(q_j, \sigma_{n-1} \dots \sigma_1)) \right) \\ &= \text{rep}_y \left(\text{sign}(i) \left(\bigcup_{j=1}^s \delta(q_i, \sigma_n, q_j) \widetilde{\delta}(q_j, \sigma_{n-1} \dots \sigma_1) \right) \right) \\ &= \text{rep}_y(\widetilde{\delta}(q_i, \sigma_n \sigma_{n-1} \dots \sigma_1)). \quad \blacksquare \end{aligned}$$

Theorem 4.3. For a fixed OFT M with input alphabet Σ , $\widetilde{M}(\Sigma^*) \in \text{NL}$.

Proof. Consider the following modification of the algorithm given at the beginning of this section which decides whether or not $y \in \widetilde{M}(\Sigma^*)$:

```

input  $y$ ;
 $\vec{v}^y := \vec{f}^y$ ;
while true do
  begin
    if  $v_1^y = |y|_{\text{eq}}$  then accept;
    guess  $\sigma \in \Sigma$ ;
     $\vec{v}^y := A^\sigma *^y \vec{v}^y$ 
  end

```

The matrices A^σ and \vec{f}^y can be kept in finite control, and the space required by \vec{v}^y is $O(\log |y|)$, since its entries are representations of strings with respect to y ; therefore, this algorithm can be implemented by a nondeterministic logspace-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, the value of v_1^y at the beginning of the while-loop after guessing x will be

$$\begin{aligned}
 v_1^y &= (A^{\sigma_n} *^y \dots *^y A^{\sigma_1} *^y \vec{f}^y)_1 \\
 &= \text{rep}_y(\widetilde{\delta}(q_1, \sigma_n \dots \sigma_1)) \\
 &= \text{rep}_y(\widetilde{M}(x)).
 \end{aligned}$$

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

$$\begin{aligned}
 v_1^y = |y|_{\text{eq}} &\iff \text{rep}_y(\widetilde{M}(x)) = |y|_{\text{eq}} \\
 &\iff \widetilde{M}(x) = y,
 \end{aligned}$$

so the algorithm accepts y if and only if $\widetilde{M}(x) = y$, for some $x \in \Sigma^*$. ■

Corollary 4.4. $\text{range}(\text{OFT}) \subseteq \text{NL}$.

Nonoptimizing versions of the OFT have been studied in the literature, particularly the deterministic and nondeterministic Generalized Sequential Machine (GSM) [5]. In a deterministic GSM M , the output $M(x)$ is the output of the unique computation of M on input x . In a nondeterministic GSM N , the output $N(x)$ is the union of the outputs of all computations on input x . It has been shown [6] that the ranges of both deterministic and nondeterministic GSMs are regular. In other words, they map regular languages to regular languages. We show that $\text{range}(\text{OFT})$ includes the regular languages, but also includes some languages which are not context-free. We denote the classes of regular and context-free languages by REG and CFL.

Theorem 4.5. $\text{REG} \subseteq \text{range}(\text{OFT})$.

Proof. Let R be a regular language accepted by DFA $M_R = (Q, \Sigma, \delta_R, q_1, F)$. Since maximization and minimization have no effect in a deterministic OFT, we build an OFT M from M_R such that $\widetilde{M}(\Sigma^*) = R$ by making every state of M_R maximizing or minimizing, and echoing the input as output. Let $M = (Q, Q_{\max}, \Sigma, \Sigma, \delta, q_1, F)$, where $Q_{\max} = Q$ (or ϕ) and δ is defined so that

$$\delta(q, \sigma, p) = \begin{cases} \sigma, & \text{if } p = \delta_R(q, \sigma); \\ \phi, & \text{if } p \neq \delta_R(q, \sigma). \end{cases}$$

We prove, by induction on $|x|$, the following claim:

$$(*) \quad \widetilde{\delta}(q, x) = \begin{cases} x, & \text{if } \delta_R(q, x) \in F; \\ \phi, & \text{if } \delta_R(q, x) \notin F. \end{cases}$$

For the empty string ϵ ,

$$\widetilde{\delta}(q, \epsilon) = \begin{cases} \epsilon, & \text{if } q \in F; \\ \phi, & \text{if } q \notin F \end{cases} = \begin{cases} \epsilon, & \text{if } \delta_R(q, \epsilon) \in F; \\ \phi, & \text{if } \delta_R(q, \epsilon) \notin F, \end{cases}$$

and for strings σx of length at least 1,

$$\begin{aligned}
\tilde{\delta}(q, \sigma x) &= \text{sign}(q) \left(\bigcup_{p \in Q} \delta(q, \sigma, p) \tilde{\delta}(p, x) \right) \\
&= \text{sign}(q) (\sigma \tilde{\delta}(\delta_R(q, \sigma), x)) \\
&= \begin{cases} \sigma x, & \text{if } \delta_R(\delta_R(q, \sigma), x) \in F; \\ \phi, & \text{if } \delta_R(\delta_R(q, \sigma), x) \notin F \end{cases} \\
&= \begin{cases} \sigma x, & \text{if } \delta_R(q, \sigma x) \in F; \\ \phi, & \text{if } \delta_R(q, \sigma x) \notin F. \end{cases}
\end{aligned}$$

If we take $q = q_1$ in (*), then

$$\tilde{M}(x) = \tilde{\delta}(q_1, x) = \begin{cases} x, & \text{if } \delta_R(q_1, x) \in F; \\ \phi, & \text{if } \delta_R(q_1, x) \notin F \end{cases} = \begin{cases} x, & \text{if } x \in R; \\ \phi, & \text{if } x \notin R; \end{cases}$$

therefore, $\tilde{M}(\Sigma^*) = R$. ■

Lemma 4.6. Let M be an OFT with input alphabet Σ and $R \subseteq \Sigma^*$ a regular language. $\tilde{M}(R) \in \text{range}(\text{OFT})$.

Proof. Suppose $M = (Q, Q_{\max}, \Sigma, \Delta, \delta, q_1, F)$ and R is accepted by DFA $M_R = (Q_R, \Sigma, \delta_R, q_R, F_R)$. Using the product construction, build an OFT M' such that $\tilde{M}'(\Sigma^*) = \tilde{M}(R)$. Let $M' = (Q', Q'_{\max}, \Sigma, \Delta, \delta', q'_1, F')$, where $Q' = Q \times Q_R$, $Q'_{\max} = Q_{\max} \times Q_R$, $q'_1 = (q_1, q_R)$, $F' = F \times F_R$, and δ' is defined so that

$$\delta'((q, s), \sigma, (p, t)) = \begin{cases} \delta(q, \sigma, p), & \text{if } t = \delta_R(s, \sigma); \\ \phi, & \text{if } t \neq \delta_R(s, \sigma). \end{cases}$$

We prove, by induction on $|x|$, the following claim:

$$(*) \quad \tilde{\delta}'((q, s), x) = \begin{cases} \tilde{\delta}(q, x), & \text{if } \delta_R(s, x) \in F_R; \\ \phi, & \text{if } \delta_R(s, x) \notin F_R. \end{cases}$$

For the empty string ϵ ,

$$\begin{aligned}
\tilde{\delta}'((q, s), \epsilon) &= \begin{cases} \epsilon, & \text{if } (q, s) \in F \times F_R; \\ \phi, & \text{if } (q, s) \notin F \times F_R \end{cases} \\
&= \begin{cases} \epsilon, & \text{if } q \in F \wedge s \in F_R; \\ \phi, & \text{if } q \notin F \wedge s \in F_R; \\ \phi, & \text{if } s \notin F_R \end{cases} \\
&= \begin{cases} \tilde{\delta}(q, \epsilon), & \text{if } s \in F_R; \\ \phi, & \text{if } s \notin F_R \end{cases} \\
&= \begin{cases} \tilde{\delta}(q, \epsilon), & \text{if } \delta_R(s, \epsilon) \in F_R; \\ \phi, & \text{if } \delta_R(s, \epsilon) \notin F_R, \end{cases}
\end{aligned}$$

and for strings σx of length at least 1,

$$\begin{aligned}
\tilde{\delta}'((q, s), \sigma x) &= \text{sign}'((q, s)) \left(\bigcup_{(p,t) \in F'} \delta'((q, s), \sigma, (p, t)) \tilde{\delta}'((p, t), x) \right) \\
&= \text{sign}(q) \left(\bigcup_{p \in Q} \delta(q, \sigma, p) \tilde{\delta}'((p, \delta_R(s, \sigma)), x) \right) \\
&= \begin{cases} \text{sign}(q) \left(\bigcup_{p \in Q} \delta(q, \sigma, p) \tilde{\delta}(p, x) \right), & \text{if } \delta_R(\delta_R(s, \sigma), x) \in F_R; \\ \phi, & \text{if } \delta_R(\delta_R(s, \sigma), x) \notin F_R \end{cases} \\
&= \begin{cases} \tilde{\delta}(q, \sigma x), & \text{if } \delta_R(s, \sigma x) \in F_R; \\ \phi, & \text{if } \delta_R(s, \sigma x) \notin F_R. \end{cases}
\end{aligned}$$

If we take $(q, s) = (q_1, q_R)$ in (*), then

$$\tilde{M}'(x) = \tilde{\delta}'((q_1, q_R), x) = \begin{cases} \tilde{\delta}(q_1, x), & \text{if } \delta_R(q_R, x) \in F_R; \\ \phi, & \text{if } \delta_R(q_R, x) \notin F_R \end{cases} = \begin{cases} \tilde{M}(x), & \text{if } x \in R; \\ \phi, & \text{if } x \notin R; \end{cases}$$

therefore, $\tilde{M}'(\Sigma^*) = \tilde{M}(R)$. ■

Theorem 4.7. $\text{range}(\text{OFT}) \not\subseteq \text{CFL}$.

Proof. Construct an OFT M with input alphabet Σ whose range $\tilde{M}(\Sigma^*)$ is not context-free. Let $M = (Q, Q_{\max}, \Sigma, \Delta, \delta, q_1, F)$, where $Q = \{q_1, q_2, q_3, q_4\}$, $Q_{\max} = Q$, $\Sigma = \Delta = \{a, b, c\} (a < b < c)$, $F = Q$, and δ is defined as pictured in the transition diagram of Figure 4.1.

We restrict the domain of M to the regular language $R = a^+b^+c^+$, using Lemma 4.6, and show that $\tilde{M}(R)$ is not context-free. Let $x = a^i b^j c^k \in R$. By construction,

Figure 4.1. An OFT whose range is not context-free

$$\begin{aligned}
\widetilde{M}(x) &= \max\{a^{3i}c^j b^{2k}, b^{2i}a^{3j}c^k, c^{2i}b^{2j}a^{2k}\} \\
&= \begin{cases} a^{3i}c^j b^{2k}, & \text{if } (3i + j + 2k > 2i + 3j + k) \wedge (3i + j + 2k > 2i + 2j + 2k); \\ b^{2i}a^{3j}c^k, & \text{if } (2i + 3j + k \geq 3i + j + 2k) \wedge (2i + 3j + k > 2i + 2j + 2k); \\ c^{2i}b^{2j}a^{2k}, & \text{if } (2i + 2j + 2k \geq 3i + j + 2k) \wedge (2i + 2j + 2k \geq 2i + 3j + k) \end{cases} \\
&= \begin{cases} a^{3i}c^j b^{2k}, & \text{if } (i + k > 2j) \wedge (i > j); \\ b^{2i}a^{3j}c^k, & \text{if } (i + k \leq 2j) \wedge (j > k); \\ c^{2i}b^{2j}a^{2k}, & \text{if } (i \leq j) \wedge (j \leq k). \end{cases}
\end{aligned}$$

Define a homomorphism $h: \{0, 1, 2\} \rightarrow \Delta^*$ by $h(0) = cc$, $h(1) = bb$, $h(2) = aa$, and consider $h^{-1}(\widetilde{M}(R) \cap c^*b^*a^*)$:

$$\begin{aligned}
h^{-1}(\widetilde{M}(R) \cap c^*b^*a^*) &= h^{-1}(\{c^{2i}b^{2j}a^{2k} \mid 1 \leq i \leq j \leq k\}) \\
&= \{0^i 1^j 2^k \mid 1 \leq i \leq j \leq k\}.
\end{aligned}$$

Since the latter is a well-known noncontext-free language, and context-freeness is preserved under inverse homomorphism and intersection with a regular language, $\widetilde{M}(R) \notin \text{CFL}$. ■

Corollary 4.8. $\text{REG} \subset \text{range}(\text{OFT})$ (REG is properly contained in range(OFT)).

5. Closure Properties

In this section, we show some closure properties of the class of OFT ranges. In particular, we show that $\text{range}(\text{OFT})$ is effectively closed under union, concatenation, Kleene closure, and monotonic homomorphism. We will see in a subsequent section that the ranges are not effectively closed under complement.

Theorem 5.1. $\text{range}(\text{OFT})$ is effectively closed under union.

Proof. Let $M' = (Q', Q'_{\max}, \Sigma', \Delta', \delta', q'_1, F')$, $M'' = (Q'', Q''_{\max}, \Sigma'', \Delta'', \delta'', q''_1, F'')$ be OFTs, and construct an OFT M having input alphabet Σ such that $\widetilde{M}(\Sigma^*) = \widetilde{M}'(\Sigma'^*) \cup \widetilde{M}''(\Sigma''^*)$. Let $M = (Q, Q_{\max}, \Sigma, \Delta, \delta, q_1, F)$, where $Q = \{q_1\} \cup Q' \cup Q''$, $Q_{\max} = \{q_1\} \cup Q'_{\max} \cup Q''_{\max}$ (or $Q'_{\max} \cup Q''_{\max}$), $\Sigma = \{\#\prime, \#\prime\prime\} \cup \Sigma' \cup \Sigma''$, $\Delta = \Delta' \cup \Delta''$, $F = F' \cup F''$, and δ is defined by

$$\delta(q, \sigma) = \begin{cases} \{(q'_1, \epsilon)\}, & \text{if } q = q_1 \wedge \sigma = \#\prime; \\ \{(q''_1, \epsilon)\}, & \text{if } q = q_1 \wedge \sigma = \#\prime\prime; \\ \delta'(q, \sigma), & \text{if } q \in Q' \wedge \sigma \in \Sigma'; \\ \delta''(q, \sigma), & \text{if } q \in Q'' \wedge \sigma \in \Sigma''; \\ \phi, & \text{otherwise.} \end{cases}$$

A transition diagram for the OFT M is pictured in Figure 5.1.

First, we observe how M operates on inputs $\#\prime x$, where $x \in \Sigma'^*$.

$$\begin{aligned} \widetilde{M}(\#\prime x) &= \widetilde{\delta}(q_1, \#\prime x) \\ &= \text{sign}(q_1) \left(\bigcup_{p \in Q} \delta(q_1, \#\prime, p) \widetilde{\delta}(p, x) \right) \\ &= \text{sign}(q_1) (\widetilde{\delta}(q'_1, x)) \\ &= \widetilde{\delta}(q'_1, x) = \widetilde{\delta}'(q'_1, x) = \widetilde{M}'(x). \end{aligned}$$

Similarly, on inputs $\#\prime\prime x$, where $x \in \Sigma''^*$, $\widetilde{M}(\#\prime\prime x) = \widetilde{M}''(x)$. If we restrict the domain of M to the regular language $R = \#\prime \Sigma'^* \cup \#\prime\prime \Sigma''^*$, using Lemma 4.6, then $\widetilde{M}(\Sigma^*) = \widetilde{M}'(\Sigma'^*) \cup \widetilde{M}''(\Sigma''^*)$. ■

Figure 5.1. Effective closure under union

Theorem 5.2. $\text{range}(\text{OFT})$ is effectively closed under concatenation.

Proof. Let $M' = (Q', Q'_{\max}, \Sigma', \Delta', \delta', q'_1, F')$, $M'' = (Q'', Q''_{\max}, \Sigma'', \Delta'', \delta'', q''_1, F'')$ be OFTs, and construct an OFT M with input alphabet Σ such that $\widetilde{M}(\Sigma^*) = \widetilde{M}'(\Sigma'^*)\widetilde{M}''(\Sigma''^*)$. Let $M = (Q, Q_{\max}, \Sigma, \Delta, \delta, q'_1, F'')$, where $Q = Q' \cup Q''$, $Q_{\max} = Q'_{\max} \cup Q''_{\max}$, $\Sigma = \{\#\} \cup \Sigma' \cup \Sigma''$, $\Delta = \Delta' \cup \Delta''$, and δ is defined by

$$\delta(q, \sigma) = \begin{cases} \delta'(q, \sigma), & \text{if } q \in Q' \wedge \sigma \in \Sigma'; \\ \delta''(q, \sigma), & \text{if } q \in Q'' \wedge \sigma \in \Sigma''; \\ \{(q''_1, \epsilon)\}, & \text{if } q \in F' \wedge \sigma = \#; \\ \phi, & \text{otherwise.} \end{cases}$$

A transition diagram for the OFT M is pictured in Figure 5.2.

Let $q \in Q'$, $x' \in \Sigma'^*$, $x'' \in \Sigma''^*$, and prove, by induction on $|x'|$, the following claim:

$$(*) \quad \widetilde{\delta}(q, x' \# x'') = \widetilde{\delta}'(q, x')\widetilde{\delta}''(q''_1, x'').$$

For the empty string ϵ ,

Figure 5.2. Effective closure under concatenation

$$\begin{aligned}
\tilde{\delta}(q, \epsilon \# x'') &= \text{sign}(q) \left(\bigcup_{p \in Q} \delta(q, \#, p) \tilde{\delta}(p, x'') \right) \\
&= \begin{cases} \text{sign}(q) (\tilde{\delta}(q_1'', x'')), & \text{if } q \in F'; \\ \phi, & \text{if } q \notin F' \end{cases} \\
&= \begin{cases} \tilde{\delta}''(q_1'', x''), & \text{if } q \in F'; \\ \phi, & \text{if } q \notin F' \end{cases} \\
&= \tilde{\delta}'(q, \epsilon) \tilde{\delta}''(q_1'', x''),
\end{aligned}$$

and for strings $\sigma x'$ of length at least 1,

$$\begin{aligned}
\tilde{\delta}(q, \sigma x' \# x'') &= \text{sign}(q) \left(\bigcup_{p \in Q} \delta(q, \sigma, p) \tilde{\delta}(p, x' \# x'') \right) \\
&= \text{sign}'(q) \left(\bigcup_{p \in Q'} \delta'(q, \sigma, p) \tilde{\delta}'(p, x') \tilde{\delta}''(q_1'', x'') \right) \\
&= \text{sign}'(q) \left(\bigcup_{p \in Q'} \delta'(q, \sigma, p) \tilde{\delta}'(p, x') \right) \tilde{\delta}''(q_1'', x'') \\
&= \tilde{\delta}'(q, \sigma x') \tilde{\delta}''(q_1'', x'').
\end{aligned}$$

If we take $q = q_1'$ in (*), then

$$\widetilde{M}(x' \# x'') = \widetilde{\delta}(q'_1, x' \# x'') = \widetilde{\delta}'(q'_1, x') \widetilde{\delta}''(q'_1, x'') = \widetilde{M}'(x') \widetilde{M}''(x'').$$

If we restrict the domain of M to the regular language $R = \Sigma'^* \# \Sigma''^*$, using Lemma 4.6, then $\widetilde{M}(\Sigma^*) = \widetilde{M}'(\Sigma'^*) \widetilde{M}''(\Sigma''^*)$. ■

Theorem 5.3. $\text{range}(\text{OFT})$ is effectively closed under Kleene closure.

Proof. Let $M' = (Q', Q'_{\max}, \Sigma', \Delta', \delta', q'_1, F')$ be an OFT, and construct an OFT M with input alphabet Σ such that $\widetilde{M}(\Sigma^*) = (\widetilde{M}'(\Sigma'^*))^*$. Let $M = (Q, Q_{\max}, \Sigma, \Delta', \delta, q_1, F)$, where $Q = \{q_1\} \cup Q'$, $Q_{\max} = \{q_1\} \cup Q'_{\max}$ (or Q'_{\max}), $\Sigma = \{\#\} \cup \Sigma'$, $F = \{q_1\} \cup F'$, and δ is defined by

$$\delta(q, \sigma) = \begin{cases} \{(q'_1, \epsilon)\}, & \text{if } q = q_1 \wedge \sigma = \#; \\ \{(q'_1, \epsilon)\}, & \text{if } q \in F' \wedge \sigma = \#; \\ \delta'(q, \sigma), & \text{if } q \in Q' \wedge \sigma \in \Sigma'; \\ \phi, & \text{otherwise.} \end{cases}$$

A transition diagram for the OFT M is pictured in Figure 5.3.

Figure 5.3. Effective Kleene closure

First, observe that $\epsilon \in \widetilde{M}(\Sigma^*)$, since $\widetilde{M}(\epsilon) = \epsilon$. Now, we show how a string in $(\widetilde{M}'(\Sigma'^*))^n$ is output by M , for $n \geq 1$. Let $q \in Q'$, $x_1, \dots, x_n \in \Sigma'^*$, and prove, by induction on n , the following claim:

$$(*) \quad \widetilde{\delta}(q, x_1 \# x_2 \# \dots \# x_n) = \widetilde{\delta}'(q, x_1) \widetilde{\delta}'(q'_1, x_2) \cdots \widetilde{\delta}'(q'_1, x_n).$$

If $n = 1$, then $\widetilde{\delta}(q, x_1) = \widetilde{\delta}'(q, x_1)$, since $\#$ does not occur in the input. If $n > 1$, we establish the claim by a second induction on $|x_1|$. For the empty string ϵ ,

$$\begin{aligned} \widetilde{\delta}(q, \epsilon \# x_2 \# \dots \# x_n) &= \text{sign}(q) \left(\bigcup_{p \in Q} \delta(q, \#, p) \widetilde{\delta}(p, x_2 \# \dots \# x_n) \right) \\ &= \begin{cases} \text{sign}(q) (\widetilde{\delta}(q'_1, x_2 \# \dots \# x_n)), & \text{if } q \in F'; \\ \phi, & \text{if } q \notin F' \end{cases} \\ &= \begin{cases} \widetilde{\delta}(q'_1, x_2 \# \dots \# x_n), & \text{if } q \in F'; \\ \phi, & \text{if } q \notin F' \end{cases} \\ &= \widetilde{\delta}'(q, \epsilon) \widetilde{\delta}(q'_1, x_2 \# \dots \# x_n) \\ &= \widetilde{\delta}'(q, \epsilon) \widetilde{\delta}'(q'_1, x_2) \cdots \widetilde{\delta}'(q'_1, x_n), \end{aligned}$$

and for strings σx_1 of length at least 1,

$$\begin{aligned} \widetilde{\delta}(q, \sigma x_1 \# x_2 \# \dots \# x_n) &= \text{sign}(q) \left(\bigcup_{p \in Q} \delta(q, \sigma, p) \widetilde{\delta}(p, x_1 \# x_2 \# \dots \# x_n) \right) \\ &= \text{sign}'(q) \left(\bigcup_{p \in Q'} \delta'(q, \sigma, p) \widetilde{\delta}'(p, x_1) \widetilde{\delta}'(q'_1, x_2) \cdots \widetilde{\delta}'(q'_1, x_n) \right) \\ &= \text{sign}'(q) \left(\bigcup_{p \in Q'} \delta'(q, \sigma, p) \widetilde{\delta}'(p, x_1) \right) \widetilde{\delta}'(q'_1, x_2) \cdots \widetilde{\delta}'(q'_1, x_n) \\ &= \widetilde{\delta}'(q, \sigma x_1) \widetilde{\delta}'(q'_1, x_2) \cdots \widetilde{\delta}'(q'_1, x_n). \end{aligned}$$

If we take $q = q'_1$ in (*), then

$$\begin{aligned} \widetilde{M}(\#x_1 \# x_2 \# \dots \# x_n) &= \widetilde{\delta}(q_1, \#x_1 \# x_2 \# \dots \# x_n) \\ &= \text{sign}(q_1) \left(\bigcup_{p \in Q} \delta(q_1, \#, p) \widetilde{\delta}(p, x_1 \# x_2 \# \dots \# x_n) \right) \\ &= \text{sign}(q_1) (\widetilde{\delta}(q'_1, x_1 \# x_2 \# \dots \# x_n)) \\ &= \widetilde{\delta}(q'_1, x_1 \# x_2 \# \dots \# x_n) \\ &= \widetilde{\delta}'(q'_1, x_1) \widetilde{\delta}'(q'_1, x_2) \cdots \widetilde{\delta}'(q'_1, x_n) \\ &= \widetilde{M}'(x_1) \widetilde{M}'(x_2) \cdots \widetilde{M}'(x_n). \end{aligned}$$

If we restrict the domain of M to the regular language $R = (\#\Sigma'^*)^*$, using Lemma 4.6, then $\widetilde{M}(\Sigma^*) = (\widetilde{M}'(\Sigma'^*))^*$. ■

Let Δ' and Δ be finite, ordered alphabets. A *homomorphism* is a function $h: \Delta'^* \rightarrow \Delta^*$ which satisfies $h(xy) = h(x)h(y)$ for every $x, y \in \Delta'^*$. A homomorphism is *monotonic* if, in addition, $x \leq y \implies h(x) \leq h(y)$. Are the homomorphic images of OFT ranges also OFT ranges? It seems that this is not the case since arbitrary homomorphisms can map different symbols to strings of different lengths, regardless of the ordering of the original symbols. We show that if the homomorphism is monotonic, then the class of OFT ranges is closed under such an operation.

Theorem 5.4. $\text{range}(\text{OFT})$ is effectively closed under monotonic homomorphism.

Proof. Let $M' = (Q', Q'_{\max}, \Sigma', \Delta', \delta', q'_1, F')$ be an OFT, $h: \Delta'^* \rightarrow \Delta^*$ a monotonic homomorphism, and construct an OFT M such that $\widetilde{M}(\Sigma^*) = h(\widetilde{M}'(\Sigma'^*))$. Let $M = (Q', Q'_{\max}, \Sigma', \Delta, \delta, q'_1, F')$, where δ is defined so that $\delta(q, \sigma, p) = h(\delta'(q, \sigma, p))$. We prove, by induction on $|x|$, the following claim:

$$(*) \quad \widetilde{\delta}(q, x) = h(\widetilde{\delta}'(q, x)).$$

For the empty string ϵ ,

$$\widetilde{\delta}(q, \epsilon) = \begin{cases} \epsilon, & \text{if } q \in F'; \\ \phi, & \text{if } q \notin F' \end{cases} = \begin{cases} h(\epsilon), & \text{if } q \in F'; \\ \phi, & \text{if } q \notin F' \end{cases} = h(\widetilde{\delta}'(q, \epsilon)),$$

and for strings σx of length at least 1,

$$\begin{aligned} \widetilde{\delta}(q, \sigma x) &= \text{sign}'(q) \left(\bigcup_{p \in Q'} \delta(q, \sigma, p) \widetilde{\delta}(p, x) \right) \\ &= \text{sign}'(q) \left(\bigcup_{p \in Q'} h(\delta'(q, \sigma, p)) h(\widetilde{\delta}'(p, x)) \right) \\ &= \text{sign}'(q) \left(\bigcup_{p \in Q'} h(\delta'(q, \sigma, p) \widetilde{\delta}'(p, x)) \right) \\ &= h \left(\text{sign}'(q) \left(\bigcup_{p \in Q'} \delta'(q, \sigma, p) \widetilde{\delta}'(p, x) \right) \right) \\ &= h(\widetilde{\delta}'(q, \sigma x)). \end{aligned}$$

If we take $q = q'_1$ in (*), then

$$\widetilde{M}(x) = \widetilde{\delta}(q'_1, x) = h(\widetilde{\delta}'(q'_1, x)) = h(\widetilde{M}'(x)),$$

and $\widetilde{M}(\Sigma'^*) = h(\widetilde{M}'(\Sigma'^*))$. ■

The applicability of closure under monotonic homomorphism is limited by the following characterization: A monotonic homomorphism h is a homomorphism for which $a < b$ implies not only $h(a) \leq h(b)$, but also $|h(a)| = |h(b)|$, for any symbols a and b . In other words, a monotonic homomorphism is simply a renaming of symbols by strings of equal length which preserves order. For example, no erasing homomorphism is monotonic unless it erases every symbol.

6. A Hierarchy in NL

In this section, we consider alternation-bounded OFTs, define a hierarchy in NL based on ranges of \max_j - and \min_j -OFTs, and present canonical languages for each level in the hierarchy.

We have shown that the range of an arbitrary OFT is in NL; therefore, the classes of ranges of \max_j - and \min_j -OFTs are trivially included in NL, for $j \geq 1$. These classes also form a hierarchy, since a \max_j -(\min_j -)OFT is trivially a \max_{j+1} -(\min_{j+1} -)OFT. We consider first a closure property which has not been discussed—intersection with regular languages. Given an OFT M with input alphabet Σ and a regular language R , is $\widetilde{M}(\Sigma^*) \cap R \in \text{range}(\text{OFT})$? We conjecture that $\widetilde{M}(\Sigma^*) \cap R$ is not necessarily in $\text{range}(\text{OFT})$; however, it is in NL, since $\widetilde{M}(\Sigma^*) \in \text{NL}$. We enrich the hierarchy within NL by closing the alternation-bounded range classes under intersection with regular languages, giving the following classes, for $j \geq 1$:

$$\begin{aligned}\max_j^{\text{OFT}} &= \{ \widetilde{M}(\Sigma^*) \cap R \mid M \text{ is a max}_j\text{-OFT} \wedge R \in \text{REG} \}, \\ \min_j^{\text{OFT}} &= \{ \widetilde{M}(\Sigma^*) \cap R \mid M \text{ is a min}_j\text{-OFT} \wedge R \in \text{REG} \}, \\ \text{OFTH} &= \bigcup_{j \geq 1} \max_j^{\text{OFT}} (= \bigcup_{j \geq 1} \min_j^{\text{OFT}}).\end{aligned}$$

Theorem 6.1.

- (1) $\text{REG} \subset \max_1^{\text{OFT}} \wedge \text{REG} \subset \min_1^{\text{OFT}}$.
- (2) $\max_j^{\text{OFT}} \cup \min_j^{\text{OFT}} \subseteq \max_{j+1}^{\text{OFT}} \cap \min_{j+1}^{\text{OFT}}$, for $j \geq 1$.
- (3) $\text{OFTH} \subseteq \text{NL}$.

Proof. (1) Let R be a regular language and construct an OFT M , as in Theorem 4.5, such that $\widetilde{M}(\Sigma^*) = R$. M can be either a \max_1 - or \min_1 -OFT (depending on whether Q_{\max} is chosen to be Q or ϕ in that construction), so $R \in \max_1^{\text{OFT}} \cap \min_1^{\text{OFT}}$. The construction of Theorem 4.7 gives us a nonregular (in fact, noncontext-free) language in \max_1^{OFT} , and a noncontext-free language in \min_1^{OFT} can be similarly obtained.

(2) It is trivial that $\max_j^{\text{OFT}} \subseteq \max_{j+1}^{\text{OFT}}$ and $\min_j^{\text{OFT}} \subseteq \min_{j+1}^{\text{OFT}}$, so we prove $\max_j^{\text{OFT}} \subseteq \min_{j+1}^{\text{OFT}}$ and $\min_j^{\text{OFT}} \subseteq \max_{j+1}^{\text{OFT}}$, by simulation. Let $M' = (Q', Q'_{\max}, \Sigma', \Delta', \delta', q'_1, F')$ be a \max_j -OFT and construct a \min_{j+1} -OFT M with input alphabet Σ such that $\widetilde{M}(\Sigma^*) = \widetilde{M}'(\Sigma'^*)$. Let $M = (Q, Q'_{\max}, \Sigma, \Delta', \delta, q_1, F')$, where $Q = \{q_1\} \cup Q'$, $\Sigma = \{\#\} \cup \Sigma'$, and δ is defined by

$$\delta(q, \sigma) = \begin{cases} \{(q'_1, \epsilon)\}, & \text{if } q = q_1 \wedge \sigma = \#; \\ \delta'(q, \sigma), & \text{if } q \in Q' \wedge \sigma \in \Sigma'; \\ \phi, & \text{otherwise.} \end{cases}$$

A transition diagram for the OFT M is pictured in Figure 6.1.

M is a \min_{j+1} -OFT, since M' is a \max_j -OFT. Let $x \in \Sigma'^*$ and consider $\widetilde{M}(\#x)$:

Figure 6.1. A \min_{j+1} -OFT having the range of a \max_j -OFT

$$\begin{aligned}
\widetilde{M}(\#x) &= \widetilde{\delta}(q_1, \#x) \\
&= \text{sign}(q_1) \left(\bigcup_{p \in Q} \delta(q_1, \#, p) \widetilde{\delta}(p, x) \right) \\
&= \text{sign}(q_1) (\widetilde{\delta}(q'_1, x)) \\
&= \widetilde{\delta}(q'_1, x) = \widetilde{\delta}'(q'_1, x) = \widetilde{M}'(x).
\end{aligned}$$

If we restrict the domain of M to the regular language $R = \#\Sigma^*$, using Lemma 4.6, then $\widetilde{M}(\Sigma^*) = \widetilde{M}'(\Sigma'^*)$. $\min_j^{\text{OFT}} \subseteq \max_{j+1}^{\text{OFT}}$ is proved similarly.

(3) Let $L \in \text{OFTH}$. $L = \widetilde{M}(\Sigma^*) \cap R$, for some \max_j - or \min_j -OFT M and regular language R ; $\widetilde{M}(\Sigma^*) \in \text{NL}$, by Theorem 4.3; and NL is closed under intersection with regular languages. ■

The inclusions of Theorem 6.1 are illustrated in Figure 6.2.

After a couple of technical lemmas, we present canonical languages in \max_j^{OFT} and \min_j^{OFT} . For $j \geq 1$, we define $\text{maxmin}_j, \text{minmax}_j: \mathcal{N}^{j+1} \rightarrow \mathcal{N}$ by

$$\begin{aligned}
\text{maxmin}_j(l_0, \dots, l_j) &= \max(l_0, \min(l_1, \max \dots (l_{j-1}, l_j) \dots)), \\
\text{minmax}_j(l_0, \dots, l_j) &= \min(l_0, \max(l_1, \min \dots (l_{j-1}, l_j) \dots)).
\end{aligned}$$

Note that $\text{maxmin}_1(l_0, l_1) = \max(l_0, l_1)$ and $\text{minmax}_1(l_0, l_1) = \min(l_0, l_1)$.

Figure 6.2. A hierarchy in NL

Lemma 6.2. For $j \geq 1$, there is a \max_j -(\min_j -)OFT M such that

$$\widetilde{M}(\#0^{l_0}\#0^{l_1} \dots \#0^{l_j}) = a^{j+1+\Sigma l_i+\max\min_j(\min\max_j)(l_0,\dots,l_j)}.$$

Proof. We construct a \max_j -OFT M , and a \min_j -OFT satisfying the claim is obtained by interchanging the maximizing and minimizing states of M . Rather than formally defining M , we give in Figure 6.3 a computation tree of M on input $x = \#0^{l_0}\#0^{l_1} \dots \#0^{l_j}$, and leave the details of construction to the reader. Some states are not explicitly shown, but states where nondeterminism occurs are shown, and the rest are implicitly configured to avoid additional alternations. The state shown as a circle may be maximizing or minimizing, depending on whether j is odd or even.

We observe that $\widetilde{M}(x) = a^{j+1+\Sigma l_i+\max\min_j(l_0,\dots,l_j)}$ by evaluating the computation tree from the bottom up. Every computation outputs an “a” for each of $j + 1$ #’s, at

Figure 6.3. Computation tree of the \max_j -OFT in Lemma 6.2

least one “a” for each of $\sum l_i$ 0’s, and an additional “a” for some $\max_{\min_j}(l_0, \dots, l_j)$ 0’s by optimization. ■

Lemma 6.3. For $j \geq 1$, there is a \min_j -(\max_j -)OFT M such that

$$\widetilde{M}(\#0^{l_0}\#0^{l_1} \dots \#0^{l_j}) = a^{j+1+\sum l_i - \max_{\min_j}(\min_{\max_j})(l_0, \dots, l_j)}.$$

Proof. We construct a \min_j -OFT M , and a \max_j -OFT satisfying the claim is obtained by interchanging the maximizing and minimizing states of M . Rather than

Figure 6.4. Computation tree of the \min_j -OFT in Lemma 6.3

formally defining M , we give in Figure 6.4 a computation tree of M on input $x = \#0^{l_0}\#0^{l_1} \dots \#0^{l_j}$:

We observe that $\widetilde{M}(x) = a^{j+1+\Sigma l_i - \max \min_j(l_0, \dots, l_j)}$ by evaluating the computation tree from the bottom up. Every computation outputs an “a” for each of $j + 1$ #’s, and an “a” for each of Σl_i 0’s, except some $\max \min_j(l_0, \dots, l_j)$ 0’s for which ϵ is output by optimization. ■

Theorem 6.4. For $j \geq 1$,

$$L_1 = \{0^k \#0^{l_0} \dots \#0^{l_j} \mid k \geq \max\min_j(l_0, \dots, l_j)\} \in \max_j^{\text{OFT}} \cap \min_j^{\text{OFT}},$$

$$L_2 = \{0^k \#0^{l_0} \dots \#0^{l_j} \mid k = \max\min_j(l_0, \dots, l_j)\} \in \max_{j+1}^{\text{OFT}} \cap \min_{j+1}^{\text{OFT}}.$$

Proof. Let $x_1 = 0^k$, $x_2 = \#0^{l_0} \dots \#0^{l_j}$.

($L_1 \in \max_j^{\text{OFT}}$) We construct a \max_j -OFT M with input alphabet $\Sigma = \{0, \#\}$ and output alphabet $\Delta = \{a, 0, \#\}$ ($a < 0 < \#$) such that $\widetilde{M}(\Sigma^*) \cap \{0, \#\}^* = L_1$. Let M' be the \max_j -OFT of Lemma 6.2, satisfying $\widetilde{M}'(x_2) = a^{j+1+\Sigma l_i + \max\min_j(l_0, \dots, l_j)}$. Construct M from M' so that its computation tree on input x_1x_2 is given by Figure 6.5.

Figure 6.5. Computation tree of a \max_j -OFT witnessing $L_1 \in \max_j^{\text{OFT}}$

If the states which are not explicitly shown are configured to avoid additional alternations, then M is a \max_j -OFT, since M' is a \max_j -OFT. By construction,

$$\begin{aligned} \widetilde{M}(x_1x_2) &= \max\{x_1x_2, \widetilde{M}'(x_2)\} \\ &= \begin{cases} x_1x_2, & \text{if } |x_1x_2| \geq |\widetilde{M}'(x_2)|; \\ \widetilde{M}'(x_2), & \text{otherwise} \end{cases} \\ &= \begin{cases} x_1x_2, & \text{if } k+j+1+\Sigma l_i \geq j+1+\Sigma l_i + \max\min_j(l_0, \dots, l_j); \\ \text{a string of a's,} & \text{otherwise} \end{cases} \\ &= \begin{cases} x_1x_2, & \text{if } k \geq \max\min_j(l_0, \dots, l_j); \\ \text{a string of a's,} & \text{otherwise.} \end{cases} \end{aligned}$$

If we restrict the domain of M to the regular language $R = 0^*(\#0^*)^{j+1}$, then $\widetilde{M}(\Sigma^*) \cap \{0, \#\}^* = L_1$.

($L_1 \in \min_j^{\text{OFT}}$) We construct a \min_j -OFT M with input alphabet $\Sigma = \{0, \#\}$ and output alphabet $\Delta = \{0, \#, b\}$ ($0 < \# < b$) such that $\widetilde{M}(\Sigma^*) \cap \{0, \#\}^* = L_1$. Let M' be the \min_j -OFT of Lemma 6.3, satisfying $\widetilde{M}'(x_2) = b^{j+1+\Sigma l_i - \max \min_j(l_0, \dots, l_j)}$. Construct M from M' so that its computation tree on input x_1x_2 is given by Figure 6.6.

Figure 6.6. Computation tree of a \min_j -OFT witnessing $L_1 \in \min_j^{\text{OFT}}$

If the states which are not explicitly shown are configured to avoid additional alternations, then M is a \min_j -OFT, since M' is a \min_j -OFT. By construction,

$$\begin{aligned}
\widetilde{M}(x_1x_2) &= \min\{x_1x_2, b^{2k}\widetilde{M}'(x_2)\} \\
&= \begin{cases} x_1x_2, & \text{if } |x_1x_2| \leq |b^{2k}\widetilde{M}'(x_2)|; \\ b^{2k}\widetilde{M}'(x_2), & \text{otherwise} \end{cases} \\
&= \begin{cases} x_1x_2, & \text{if } k+j+1+\Sigma l_i \leq 2k+j+1+\Sigma l_i - \max \min_j(l_0, \dots, l_j); \\ \text{a string of b's,} & \text{otherwise} \end{cases} \\
&= \begin{cases} x_1x_2, & \text{if } k \geq \max \min_j(l_0, \dots, l_j); \\ \text{a string of b's,} & \text{otherwise.} \end{cases}
\end{aligned}$$

If we restrict the domain of M to the regular language $R = 0^*(\#0^*)^{j+1}$, then $\widetilde{M}(\Sigma^*) \cap \{0, \#\}^* = L_1$.

($L_2 \in \max_{j+1}^{\text{OFT}}$) We construct a \max_{j+1} -OFT M with input alphabet $\Sigma = \{0, \#\}$ and output alphabet $\Delta = \{a, 0, \#\}$ ($a < 0 < \#$) such that $\widetilde{M}(\Sigma^*) \cap \{0, \#\}^* = L_2$. Let M' be the \max_j -OFT of Lemma 6.2, satisfying $\widetilde{M}'(x_2) = a^{j+1+\Sigma l_i + \max \min_j(l_0, \dots, l_j)}$, and let M'' be the \min_j -OFT of Lemma 6.3, satisfying $\widetilde{M}''(x_2) = a^{j+1+\Sigma l_i - \max \min_j(l_0, \dots, l_j)}$. Construct M from M' and M'' so that its computation tree on input $x_1 x_2$ is given by Figure 6.7.

Figure 6.7. Computation tree of a \max_{j+1} -OFT witnessing $L_2 \in \max_{j+1}^{\text{OFT}}$

If the states which are not explicitly shown are configured to avoid additional alternations, then M is a \max_{j+1} -OFT, since M'' is a \min_j -OFT. By construction,

$$\begin{aligned}
\widetilde{M}(x_1x_2) &= \max\{x_1x_2, \widetilde{M}'(x_2), a^{2k}\widetilde{M}''(x_2)\} \\
&= \begin{cases} x_1x_2, & \text{if } (|x_1x_2| \geq |\widetilde{M}'(x_2)|) \wedge (|x_1x_2| \geq |a^{2k}\widetilde{M}''(x_2)|); \\ \text{a string of a's,} & \text{otherwise} \end{cases} \\
&= \begin{cases} x_1x_2, & \text{if } (k+j+1+\Sigma l_i \geq j+1+\Sigma l_i + \max\min_j(l_0, \dots, l_j)) \wedge \\ & (k+j+1+\Sigma l_i \geq 2k+j+1+\Sigma l_i - \max\min_j(l_0, \dots, l_j)); \\ \text{a string of a's,} & \text{otherwise} \end{cases} \\
&= \begin{cases} x_1x_2, & \text{if } k \geq \max\min_j(l_0, \dots, l_j) \wedge k \leq \max\min_j(l_0, \dots, l_j); \\ \text{a string of a's,} & \text{otherwise} \end{cases} \\
&= \begin{cases} x_1x_2, & \text{if } k = \max\min_j(l_0, \dots, l_j); \\ \text{a string of a's,} & \text{otherwise.} \end{cases}
\end{aligned}$$

If we restrict the domain of M to the regular language $R = 0^*(\#0^*)^{j+1}$, then $\widetilde{M}(\Sigma^*) \cap \{0, \#\}^* = L_2$.

($L_2 \in \min_{j+1}^{\text{OFT}}$) We construct a \min_{j+1} -OFT M with input alphabet $\Sigma = \{0, \#\}$ and output alphabet $\Delta = \{0, \#, b\}$ ($0 < \# < b$) such that $\widetilde{M}(\Sigma^*) \cap \{0, \#\}^* = L_2$. Let M' be the \max_j -OFT of Lemma 6.2, satisfying $\widetilde{M}'(x_2) = b^{j+1+\Sigma l_i + \max\min_j(l_0, \dots, l_j)}$, and let M'' be the \min_j -OFT of Lemma 6.3, satisfying $\widetilde{M}''(x_2) = b^{j+1+\Sigma l_i - \max\min_j(l_0, \dots, l_j)}$. Construct M from M' and M'' so that its computation tree on input x_1x_2 is given by Figure 6.8.

Figure 6.8. Computation tree of a \min_{j+1} -OFT witnessing $L_2 \in \min_{j+1}^{\text{OFT}}$

If the states which are not explicitly shown are configured to avoid additional alternations, then M is a \min_{j+1} -OFT, since M' is a \max_j -OFT. By construction,

$$\begin{aligned}
\widetilde{M}(x_1x_2) &= \min\{x_1x_2, \widetilde{M}'(x_2), b^{2k}\widetilde{M}''(x_2)\} \\
&= \begin{cases} x_1x_2, & \text{if } (|x_1x_2| \leq |\widetilde{M}'(x_2)|) \wedge (|x_1x_2| \leq |b^{2k}\widetilde{M}''(x_2)|); \\ \text{a string of b's,} & \text{otherwise} \end{cases} \\
&= \begin{cases} x_1x_2, & \text{if } (k+j+1+\sum l_i \leq j+1+\sum l_i + \max \min_j(l_0, \dots, l_j)) \wedge \\ & (k+j+1+\sum l_i \leq 2k+j+1+\sum l_i - \max \min_j(l_0, \dots, l_j)); \\ \text{a string of b's,} & \text{otherwise} \end{cases} \\
&= \begin{cases} x_1x_2, & \text{if } k \leq \max \min_j(l_0, \dots, l_j) \wedge k \geq \max \min_j(l_0, \dots, l_j); \\ \text{a string of b's,} & \text{otherwise} \end{cases} \\
&= \begin{cases} x_1x_2, & \text{if } k = \max \min_j(l_0, \dots, l_j); \\ \text{a string of b's,} & \text{otherwise.} \end{cases}
\end{aligned}$$

If we restrict the domain of M to the regular language $R = 0^*(\#0^*)^{j+1}$, then $\widetilde{M}(\Sigma^*) \cap \{0, \#\}^* = L_2$. ■

There are several open questions in conjunction with the hierarchy that we have presented. Which inclusions pictured in Figure 6.2 are proper? What interesting natural problems are in the hierarchy? Can some kind of pumping lemma be used to distinguish the levels? The canonical languages we give are candidates which might lie properly between the levels of the hierarchy, but we are unable to distinguish (and have no candidates which might separate) \max_j^{OFT} and \min_j^{OFT} , for any $j \geq 1$. The hierarchy and the open questions surrounding it remain for future work, and we hope to resolve some of these issues.

7. Decision Problems

In this section, we identify the complexity of several decision problems involving OFTs. Among these are questions about their ranges and optimal output functions. In all cases, we establish complexity-theoretic lower bounds; and in some cases, we pinpoint the complexity by showing completeness for some complexity class. We prove that the range inequivalence problem is undecidable, and conclude that $\text{range}(\text{OFT})$ is not effectively closed under complement and is properly included in NL.

We consider the following decision problems:

Range Nonemptiness

Instance: M , an OFT with input alphabet Σ .

Question: $\widetilde{M}(\Sigma^*) \neq \phi$?

Range Membership

Instance: M , an OFT with input alphabet Σ and output alphabet Δ ; $y \in \Delta^*$.

Question: $y \in \widetilde{M}(\Sigma^*)$?

Range Nonuniversality

Instance: M , an OFT with input alphabet Σ and output alphabet Δ .

Question: $\widetilde{M}(\Sigma^*) \neq \Delta^*$?

Length Inequivalence

Instance: M', M'' , OFTs with input alphabet Σ .

Question: Is there an $x \in \Sigma^*$ such that $|\widetilde{M}'(x)| \neq |\widetilde{M}''(x)|$?

Nonunary Range

Instance: M , an OFT with input alphabet Σ and output alphabet $\Delta \supseteq \{\#\}$.

Question: $\widetilde{M}(\Sigma^*) \not\subseteq \#^*$?

Range Nonregularity

Instance: M , an OFT with input alphabet Σ .

Question: Is $\widetilde{M}(\Sigma^*)$ not regular?

Bounded Length Inequivalence

Instance: M', M'' , OFTs with input alphabet Σ ; $z \in \Sigma^*$.

Question: Is there an $x \in \Sigma^*$ such that $|x| \leq |z|$ and $|\widetilde{M}'(x)| \neq |\widetilde{M}''(x)|$?

Bounded Nonunary Range

Instance: M , an OFT with input alphabet Σ and output alphabet $\Delta \supseteq \{\#\}$; $z \in \Sigma^*$.

Question: Is there an $x \in \Sigma^*$ such that $|x| \leq |z|$ and $\widetilde{M}(x) \not\subseteq \#^*$?

Inequivalence

Instance: M', M'' , OFTs with input alphabet Σ .

Question: Is there an $x \in \Sigma^*$ such that $\widetilde{M}'(x) \neq \widetilde{M}''(x)$?

Range Inequivalence

Instance: M', M'' , OFTs with input alphabets Σ', Σ'' .

Question: $\widetilde{M}'(\Sigma'^*) \neq \widetilde{M}''(\Sigma''^*)$?

Our measure of relative complexity is logspace reducibility, which we denote \leq_m^L . Hardness and completeness results for complexity classes are given with respect to \leq_m^L -reducibility. The classes of recursive and recursively enumerable languages are denoted REC and RE. We prove hardness by reduction from the following problems of known complexity:

Graph Accessibility Problem (GAP)

Instance: $G = (V, E)$, a directed graph with vertex set V and edge set $E \subseteq V \times V$; $v_1, v_n \in V$.

Question: Is there a path in G from v_1 to v_n ?

3-Partition

Instance: $A = \{a_1, \dots, a_{3m}\}$, a finite set; $s: A \rightarrow \mathcal{Z}^+$, a size function; and $B \in \mathcal{Z}^+$, a bound, such that

$$(1) \quad B/4 < s(a) < B/2, \quad \text{for } a \in A;$$

$$(2) \quad \bigcup_{a \in A} s(a) = mB.$$

Question: Is there a partition of A into m disjoint 3-sets S_1, \dots, S_m , such that for $1 \leq i \leq m$,

$$\sum_{a \in S_i} s(a) = B?$$

NFA Nonuniversality

Instance: N , an NFA with input alphabet Σ .

Question: $L(N) \neq \Sigma^*$?

NFA Inequivalence

Instance: N', N'' , NFAs with input alphabet Σ .

Question: $L(N') \neq L(N'')$?

Post's Correspondence Problem (PCP)

Instance: f, g , homomorphisms from Σ to Δ^+ .

Question: Is there an $x \in \Sigma^+$ such that $f(x) = g(x)$?

Theorem 7.1.

- (1) GAP is NL-complete.
- (2) 3-Partition is strongly NP-complete.
- (3) NFA Nonuniversality is PSPACE-complete.
- (4) NFA Inequivalence is PSPACE-complete.
- (5) PCP is RE-complete.

Proof. (1) GAP was proved NL-complete by Jones [12], based on the “threadable mazes” of Savitch [16]. (2) 3-Partition was first proved NP-complete by Garey and Johnson [3], and shown to be strongly NP-complete by the same authors [4]. (3,4) PSPACE-completeness of NFA Nonuniversality and Inequivalence follow by a simple reduction from the same problems for regular expressions, which were proved PSPACE-complete by Stockmeyer and Meyer [20] and Stockmeyer [18]. (5) PCP was proved undecidable by Post [15], and was also studied by Floyd [2]. ■

When providing \leq_m^L -reductions from these problems, we omit arguments that the construction can be achieved using logspace, since they are generally straightforward. Occasionally, we observe that the construction causes at most a polynomial increase in the size of the instances, and claim sufficiency.

Theorem 7.2.

- (1) Range Nonemptiness is NL-complete.
- (2) Range Membership is PSPACE-complete.
- (3) Range Nonuniversality is PSPACE-hard.
- (4) Length Inequivalence is PSPACE-hard.
- (5) Nonunary Range is PSPACE-hard.
- (6) Range Nonregularity is PSPACE-hard.
- (7) Bounded Length Inequivalence is NP-complete.
- (8) Bounded Nonunary Range is NP-complete.
- (9) Inequivalence is RE-complete.
- (10) Range Inequivalence is RE-complete.

Proof. (1) First, we show membership in NL. Consider the following algorithm which decides whether or not $\widetilde{M}(\Sigma^*) \neq \phi$:

```

input  $M = (Q, Q_{\max}, \Sigma, \Delta, \delta, q_1, F)$ ;
 $p := q_1$ ;
while true do
  begin
    if  $p \in F$  then accept;
    guess  $\sigma \in \Sigma$ ;
    if  $\delta(p, \sigma) = \phi$  then reject;
    guess  $(p, w) \in \delta(p, \sigma)$ 
  end

```

The space required to store p and σ is logarithmic in $\|Q\|$ and $\|\Sigma\|$; therefore, this algorithm can be implemented by a nondeterministic logspace-bounded Turing machine. Next, we show hardness by \leq_m^L -reduction from GAP. Let $G = (V, E)$, where $V = \{v_1, \dots, v_n\}$, $E \subseteq V \times V$. We construct an OFT M with input alphabet Σ such that $\widetilde{M}(\Sigma^*) \neq \phi$ if and only if there is a path in G from v_1 to v_n . Let $M = (V, V, \Sigma, \Delta, \delta, v_1, \{v_n\})$, where $\Sigma = \Delta = \{0\}$, and $\delta(v_i, 0) = \{(v_j, 0) \mid (v_i, v_j) \in E\}$. $\widetilde{M}(\Sigma^*) \neq \phi$ if and only if there is an accepting computation of M (from initial state to final state), if and only if there is a path in G from v_1 to v_n .

(2) We have shown in Theorem 4.3 that membership can be decided nondeterministically using space $O(\|Q\| \log |y|)$, where $\|Q\|$ is the number of states in M . By Savitch's Theorem, Range Membership \in NSPACE($n \log n$) \subseteq PSPACE. We show hardness by \leq_m^L -reduction from NFA Inequivalence. Let $N' = (Q', \Sigma', \delta', q'_1, F')$, $N'' = (Q'', \Sigma', \delta'', q''_1, F'')$ be NFAs, and construct an OFT M with input alphabet Σ and output alphabet $\Delta = \{a\}$ such that $a \in \widetilde{M}(\Sigma^*)$ if and only if $L(N') \neq L(N'')$. Assume, without loss of generality, that $\delta'(q', \sigma) \neq \phi$ and $\delta''(q'', \sigma) \neq \phi$, for $q' \in Q'$, $q'' \in Q''$, and $\sigma \in \Sigma'$. Let $M = (Q, Q_{\max}, \Sigma, \Delta, \delta, q_1, F)$, where $Q = Q' \times Q'' \cup \{q_f\}$,

$Q_{\max} = Q' \times Q''$ (or $Q' \times Q'' \cup \{q_f\}$), $\Sigma = \{\#\} \cup \Sigma'$, $\Delta = \{a\}$, $q_1 = (q'_1, q''_1)$, $F = \{q_f\}$,

and δ is defined by

$$\delta((q', q''), \sigma) = \begin{cases} (\delta'(q', \sigma) \times \delta''(q'', \sigma)) \times \{\epsilon\}, & \text{if } \sigma \in \Sigma'; \\ \{(q_f, \epsilon)\}, & \text{if } q' \notin F' \wedge q'' \notin F'' \wedge \sigma = \#; \\ \{(q_f, aa)\}, & \text{if } q' \in F' \wedge q'' \in F'' \wedge \sigma = \#; \\ \{(q_f, a)\}, & \text{if } (q' \in F' \oplus q'' \in F'') \wedge \sigma = \#, \end{cases}$$

$$\delta(q_f, \sigma) = \phi.$$

Let $q' \in Q'$, $q'' \in Q''$, $x \in \Sigma'^*$, and prove, by induction on $|x|$, the following claim:

$$(*) \quad \tilde{\delta}((q', q''), x\#) = \begin{cases} \epsilon, & \text{if } \delta'(q', x) \cap F' = \phi \wedge \delta''(q'', x) \cap F'' = \phi; \\ aa, & \text{if } \delta'(q', x) \cap F' \neq \phi \wedge \delta''(q'', x) \cap F'' \neq \phi; \\ a, & \text{otherwise.} \end{cases}$$

For the empty string ϵ ,

$$\begin{aligned} \tilde{\delta}((q', q''), \#) &= \max\left(\bigcup_{p \in Q} \delta((q', q''), \#, p) \tilde{\delta}(p, \epsilon)\right) \\ &= \max\left(\delta((q', q''), \#, q_f) \tilde{\delta}(q_f, \epsilon)\right) \\ &= \delta((q', q''), \#, q_f) \\ &= \begin{cases} \epsilon, & \text{if } q' \notin F' \wedge q'' \notin F''; \\ aa, & \text{if } q' \in F' \wedge q'' \in F''; \\ a, & \text{otherwise} \end{cases} \\ &= \begin{cases} \epsilon, & \text{if } \delta'(q', \epsilon) \cap F' = \phi \wedge \delta''(q'', \epsilon) \cap F'' = \phi; \\ aa, & \text{if } \delta'(q', \epsilon) \cap F' \neq \phi \wedge \delta''(q'', \epsilon) \cap F'' \neq \phi; \\ a, & \text{otherwise,} \end{cases} \end{aligned}$$

and for strings σx of length at least 1,

$$\begin{aligned} \tilde{\delta}((q', q''), \sigma x\#) &= \max\left(\bigcup_{p \in Q} \delta((q', q''), \sigma, p) \tilde{\delta}(p, x\#)\right) \\ &= \max\left(\bigcup_{(p', p'') \in \delta'(q', \sigma) \times \delta''(q'', \sigma)} \tilde{\delta}((p', p''), x\#)\right) \\ &= \begin{cases} \epsilon, & \text{if } \forall (p', p'') \in \delta'(q', \sigma) \times \delta''(q'', \sigma), \tilde{\delta}((p', p''), x\#) = \epsilon; \\ aa, & \text{if } \exists (p', p'') \in \delta'(q', \sigma) \times \delta''(q'', \sigma), \tilde{\delta}((p', p''), x\#) = aa; \\ a, & \text{otherwise} \end{cases} \\ &= \begin{cases} \epsilon, & \text{if } \delta'(\delta'(q', \sigma), x\#) \cap F' = \phi \wedge \delta''(\delta''(q'', \sigma), x\#) \cap F'' = \phi; \\ aa, & \text{if } \delta'(\delta'(q', \sigma), x\#) \cap F' \neq \phi \wedge \delta''(\delta''(q'', \sigma), x\#) \cap F'' \neq \phi; \\ a, & \text{otherwise} \end{cases} \\ &= \begin{cases} \epsilon, & \text{if } \delta'(q', \sigma x\#) \cap F' = \phi \wedge \delta''(q'', \sigma x\#) \cap F'' = \phi; \\ aa, & \text{if } \delta'(q', \sigma x\#) \cap F' \neq \phi \wedge \delta''(q'', \sigma x\#) \cap F'' \neq \phi; \\ a, & \text{otherwise.} \end{cases} \end{aligned}$$

If we take $(q', q'') = (q'_1, q''_1)$ in (*), then

$$\begin{aligned}
a \in \widetilde{M}(\Sigma^*) &\iff a \in \widetilde{M}(\Sigma'^* \#) \\
&\iff \widetilde{M}(x\#) = a, \quad \text{some } x \in \Sigma'^* \\
&\iff \widetilde{\delta}((q'_1, q''_1), x\#) = a, \quad \text{some } x \in \Sigma'^* \\
&\iff \delta'(q'_1, x) \cap F' \neq \phi \oplus \delta''(q''_1, x) \cap F'' \neq \phi, \quad \text{some } x \in \Sigma'^* \\
&\iff x \in L(N') \oplus x \in L(N''), \quad \text{some } x \in \Sigma'^* \\
&\iff L(N') \neq L(N'').
\end{aligned}$$

(3) We show hardness by \leq_m^L -reduction from NFA Nonuniversality. Let $N = (Q', \Sigma', \delta', q'_1, F')$ be an NFA, and construct an OFT M having Σ' as its input and output alphabets such that $\widetilde{M}(\Sigma'^*) = L(N)$. It follows that $\widetilde{M}(\Sigma'^*) \neq \Sigma'^*$ if and only if $L(N) \neq \Sigma'^*$. Let $M = (Q', Q', \Sigma', \Sigma', \delta, q'_1, F')$, where δ is defined by $\delta(q, \sigma) = \delta'(q, \sigma) \times \{\sigma\}$. An induction on $|x|$ proves the following claim:

$$(*) \quad \widetilde{\delta}(q, x) = \begin{cases} x, & \text{if } \delta'(q, x) \cap F' \neq \phi; \\ \phi, & \text{if } \delta'(q, x) \cap F' = \phi. \end{cases}$$

If we take $q = q'_1$ in (*), then

$$\widetilde{M}(x) = \widetilde{\delta}(q'_1, x) = \begin{cases} x, & \text{if } \delta'(q'_1, x) \cap F' \neq \phi; \\ \phi, & \text{if } \delta'(q'_1, x) \cap F' = \phi \end{cases} = \begin{cases} x, & \text{if } x \in L(N); \\ \phi, & \text{if } x \notin L(N); \end{cases}$$

therefore, $\widetilde{M}(\Sigma'^*) = L(N)$.

(4) We show hardness by \leq_m^L -reduction from NFA Inequivalence. Let N', N'' be NFAs with input alphabet Σ , and construct OFTs M', M'' , as in Theorem 7.2(3). Recall that for $x \in \Sigma^*$,

$$\widetilde{M}'(x) = \begin{cases} x, & \text{if } x \in L(N'); \\ \phi, & \text{if } x \notin L(N'), \end{cases} \quad \widetilde{M}''(x) = \begin{cases} x, & \text{if } x \in L(N''); \\ \phi, & \text{if } x \notin L(N''). \end{cases}$$

It follows that

$$|\widetilde{M}'(x)| = \begin{cases} |x|, & \text{if } x \in L(N'); \\ \phi, & \text{if } x \notin L(N'), \end{cases} \quad |\widetilde{M}''(x)| = \begin{cases} |x|, & \text{if } x \in L(N''); \\ \phi, & \text{if } x \notin L(N''); \end{cases}$$

therefore, there is an $x \in \Sigma^*$ such that $|\widetilde{M}'(x)| \neq |\widetilde{M}''(x)|$ if and only if $L(N') \neq L(N'')$.

We conjecture that Range Nonuniversality and Length Inequivalence are properly harder than PSPACE—perhaps undecidable—since optimality is not used in any essential way in the reductions of Theorems 7.2(3) and 7.2(4).

(5) We show hardness by \leq_m^L -reduction from Length Inequivalence. Let M', M'' be OFTs with input alphabet Σ' , and construct an OFT M with input alphabet Σ and output alphabet $\Delta \supseteq \{\#\}$, such that $\widetilde{M}(\Sigma^*) \not\subseteq \#^*$ if and only if there is an $x \in \Sigma'^*$ such that $|\widetilde{M}'(x)| \neq |\widetilde{M}''(x)|$. Let Δ', Δ'' be the output alphabets of M', M'' . Define monotonic homomorphisms $h': \Delta'^* \rightarrow \#^*$, $h'': \Delta''^* \rightarrow \#^*$ by $h'(a') = h''(a'') = \#$, and construct, using Theorem 5.4, OFTs $M'_{\#}, M''_{\#}$ such that for $x \in \Sigma'^*$, $\widetilde{M}'_{\#}(x) = h'(\widetilde{M}'(x))$ and $\widetilde{M}''_{\#}(x) = h''(\widetilde{M}''(x))$. $M'_{\#}$ and $M''_{\#}$ are exactly like M' and M'' , except they convert every output symbol to $\#$. Let $\Sigma = \{\#\} \cup \Sigma'$, $\Delta = \{\#\} \cup \Delta' \cup \Delta''$ ($\Delta', \Delta'' < \#$), and fix $a' \in \Delta', a'' \in \Delta''$. Rather than formally defining M , we give in Figure 7.1 a computation tree of M on input $\#\#x$, where $x \in \Sigma'^*$, and leave the details of construction to the reader.

By construction,

$$\begin{aligned} \widetilde{M}(\#\#x) &= \min(\max\{a'\widetilde{M}'(x), \#\widetilde{M}''_{\#}(x)\} \cup \max\{a''\widetilde{M}''(x), \#\widetilde{M}'_{\#}(x)\}) \\ &= \begin{cases} \min\{a'\widetilde{M}'(x), \#\widetilde{M}'_{\#}(x)\}, & \text{if } |\widetilde{M}'(x)| > |\widetilde{M}''(x)|; \\ \min\{\#\widetilde{M}''_{\#}(x), a''\widetilde{M}''(x)\}, & \text{if } |\widetilde{M}'(x)| < |\widetilde{M}''(x)|; \\ \min\{\#\widetilde{M}''_{\#}(x), \#\widetilde{M}'_{\#}(x)\}, & \text{if } |\widetilde{M}'(x)| = |\widetilde{M}''(x)| \end{cases} \\ &= \begin{cases} a'\widetilde{M}'(x), & \text{if } |\widetilde{M}'(x)| > |\widetilde{M}''(x)|; \\ a''\widetilde{M}''(x), & \text{if } |\widetilde{M}'(x)| < |\widetilde{M}''(x)|; \\ \#\widetilde{M}'_{\#}(x) (= \#\widetilde{M}''_{\#}(x)), & \text{if } |\widetilde{M}'(x)| = |\widetilde{M}''(x)|. \end{cases} \end{aligned}$$

If we restrict the domain of M to the regular language $R = \#\#\Sigma'^*$, then $\widetilde{M}(\Sigma^*) \not\subseteq \#^*$ if and only if there is an $x \in \Sigma'^*$ such that $|\widetilde{M}'(x)| \neq |\widetilde{M}''(x)|$.

Figure 7.1. Reduction from Length Inequivalence to Nonunary Range

(6) We show hardness by \leq_m^L -reduction from Range Nonuniversality. Let M' be an OFT with input alphabet Σ' and output alphabet Δ' . Let M'' be an OFT with input alphabet Σ' and output alphabet Δ' such that $\widetilde{M}''(\Sigma'^*)$ is not regular, for example, as in Theorem 4.7. Construct, using Theorems 4.5, 5.1, and 5.2, an OFT M with input alphabet Σ and output alphabet $\Delta = \{\#\} \cup \Delta'$ such that

$$\widetilde{M}(\Sigma^*) = \widetilde{M}''(\Sigma'^*)\#\Delta'^* \cup \Delta'^*\#\widetilde{M}'(\Sigma'^*).$$

We show that $\widetilde{M}(\Sigma^*)$ is nonregular if and only if $\widetilde{M}'(\Sigma'^*) \neq \Delta'^*$. Suppose $\widetilde{M}'(\Sigma'^*) \neq \Delta'^*$, and let $y \in \Delta'^* - \widetilde{M}'(\Sigma'^*)$. $\widetilde{M}(\Sigma^*)/y = \widetilde{M}''(\Sigma'^*)$ is not regular; therefore, $\widetilde{M}(\Sigma^*)$ is not regular, since regularity is preserved under quotient with y . Conversely, suppose $\widetilde{M}'(\Sigma'^*) = \Delta'^*$. In that case, $\widetilde{M}(\Sigma^*) = \Delta'^*\#\Delta'^*$ is regular.

Note that Theorem 7.2(6) is an application of a “resource-bounded” Greibach’s Theorem, a technique which was used by Hunt and Rosenkrantz [9] to prove PSPACE-hardness results for some decision problems involving regular expressions.

(7) First, we show membership in NP by applying the bounded quantifier characterization of the polynomial hierarchy given by Stockmeyer and Meyer [20]. The existentially quantified variable x has length bounded by a polynomial ($|z|$) in the size of the instance, and the predicate $|\widetilde{M}'(x)| \neq |\widetilde{M}''(x)|$ is polynomially decidable using Corollary 3.2; therefore, Bounded Length Inequivalence is in NP. Next, we show hardness by \leq_m^L -reduction from 3-Partition. Let $A = \{a_1, \dots, a_{3m}\}$, $s: A \rightarrow \mathcal{Z}^+$, $B \in \mathcal{Z}^+$ be an instance of 3-Partition. We require the sizes $s(a)$ and bound B to be given in unary; however, since 3-Partition is strongly NP-complete, this “unary” version remains NP-complete. Let $z = \#^{3m+1}$, and construct OFTs M' , M'' with input alphabet $\Sigma = \{\#, s_1, \dots, s_m\}$ and output alphabet $\Delta = \{\#\}$ such that there is an $x \in \Sigma^*$ satisfying $|x| \leq |z|$ and $|\widetilde{M}'(x)| \neq |\widetilde{M}''(x)|$ if and only if there is a 3-partition of A . Let M' , M'' be defined by the transition diagrams in Figure 7.2.

Consider $R \subseteq \Sigma^*$ defined by $R = \{x \in \Sigma^{3m} \mid s_1, \dots, s_m \text{ each occur 3 times in } x\}$. R is a regular language whose strings x specify a partition of A into 3-sets defined by

$$S_i^x = \{a_j \mid 1 \leq j \leq m \wedge \text{the } j^{\text{th}} \text{ symbol of } x \text{ is } s_i\}.$$

Let $x \in R$ and consider $|\widetilde{M}'(\#x)|$ and $|\widetilde{M}''(\#x)|$. By construction,

$$\begin{aligned} |\widetilde{M}'(\#x)| &= 1 + \sum_{a \in A} s(a) = 1 + mB, \\ |\widetilde{M}''(\#x)| &= \min\left(\max\left(\sum_{a \in S_1^x} ms(a), \dots, \sum_{a \in S_m^x} ms(a)\right), 1 + \sum_{a \in A} s(a)\right) \\ &= \min\left(\max\left(m \sum_{a \in S_1^x} s(a), \dots, m \sum_{a \in S_m^x} s(a)\right), 1 + mB\right). \end{aligned}$$

Suppose there is a 3-partition of A into 3-sets S_1^x, \dots, S_m^x specified by some $x \in R$. It follows that $|\#x| = 3m + 1 = |z|$ and

Figure 7.2. Reduction from 3-Partition to Bounded Length Inequivalence

$$\begin{aligned}
 |\widetilde{M}''(\#x)| &= \min\left(\max\left(m \sum_{a \in S_1^x} s(a), \dots, m \sum_{a \in S_m^x} s(a)\right), 1 + mB\right) \\
 &= \min(\max(mB, \dots, mB), 1 + mB) \\
 &= mB \neq 1 + mB = |\widetilde{M}'(\#x)|.
 \end{aligned}$$

Conversely, suppose there is no 3-partition of A . Then, for $x \in R$,

$$\max\left(\sum_{a \in S_1^x} s(a), \dots, \sum_{a \in S_m^x} s(a)\right) > B,$$

and

$$\begin{aligned} |\widetilde{M}''(\#x)| &= \min\left(\max\left(m \sum_{a \in S_1^x} s(a), \dots, m \sum_{a \in S_m^x} s(a)\right), 1 + mB\right) \\ &= 1 + mB = |\widetilde{M}'(\#x)|. \end{aligned}$$

If we restrict the domains of M' , M'' to the regular language $\#R$, then there is an $x \in \Sigma^*$ such that $|x| \leq |z|$ and $|\widetilde{M}'(x)| \neq |\widetilde{M}''(x)|$ if and only if there is a 3-partition of A . We note that since the sizes $s(a)$ and bound B were given in unary, the number of states and lengths of transition outputs in M' and M'' are polynomially bounded in the size of the instance of 3-Partition. Hence, this reduction can be achieved in logspace.

(8) First, we show membership in NP. The existentially quantified variable x has length bounded by a polynomial ($|z|$) in the size of the instance, and the predicate $\widetilde{M}(x) \not\subseteq \#^*$ is polynomially decidable using Corollary 3.2; therefore, Bounded Nonunary Range is in NP. Next, we show hardness by \leq_m^L -reduction from Bounded Length Inequivalence. Let M' , M'' be OFTs with input alphabet Σ' , $z' \in \Sigma'^*$. We let $z = \#^{|z'|+2}$, and construct an OFT M with input alphabet Σ and output alphabet $\Delta \supseteq \{\#\}$ such that there is an $x \in \Sigma^*$ satisfying $\widetilde{M}(x) \not\subseteq \#^*$ if and only if there is an $x' \in \Sigma'^*$ satisfying $|\widetilde{M}'(x')| \neq |\widetilde{M}''(x')|$. Let M be constructed from M' , M'' as in Theorem 7.2(5) (see Figure 7.1). Recall that for $x' \in \Sigma'^*$, $\widetilde{M}(\#\#x') \not\subseteq \#^*$ if and only if $|\widetilde{M}'(x')| \neq |\widetilde{M}''(x')|$. Since $|\#\#x'| = |x'| + 2 \leq |z'| + 2 = |z|$ if and only if $|x'| \leq |z'|$, there is an $x \in \Sigma^*$ such that $|x| \leq |z|$ and $\widetilde{M}(x) \not\subseteq \#^*$ if and only if there is an $x' \in \Sigma'^*$ such that $|x'| \leq |z'|$ and $|\widetilde{M}'(x')| \neq |\widetilde{M}''(x')|$.

(9) Inequivalence is recursively enumerable, since the existentially quantified predicate $\widetilde{M}'(x) \neq \widetilde{M}''(x)$ is decidable using the algorithm of Corollary 3.2. Next, we show hardness by \leq_m^L -reduction from PCP. Let f, g be homomorphisms from Σ to Δ^+ . We construct OFTs M', M'' with input alphabet $\Sigma' = \{\#\} \cup \Sigma$ and output alphabet $\Delta' = \{a, b\} \cup \Delta$ ($\Delta < a < b$) such that there is an $x' \in \Sigma'^*$ satisfying $\widetilde{M}'(x') \neq \widetilde{M}''(x')$

Figure 7.3. Reduction from PCP to Inequivalence

if and only if there is an $x \in \Sigma^+$ satisfying $f(x) = g(x)$. Let M' , M'' be defined by the transition diagrams in Figure 7.3.

For $x \in \Sigma^+$, we have by construction,

$$\begin{aligned}
 \widetilde{M}'(\#x\#) &= \max\{f(x)a, g(x)a\} \\
 &= \begin{cases} f(x)a, & \text{if } f(x) > g(x); \\ f(x)a(= g(x)a), & \text{if } f(x) = g(x); \\ g(x)a, & \text{if } f(x) < g(x), \end{cases} \\
 \widetilde{M}''(\#x\#) &= \max(\max\{f(x)a, g(x)a\} \cup \min\{f(x)b, g(x)b\}) \\
 &= \begin{cases} \max\{f(x)a, g(x)b\}, & \text{if } f(x) \geq g(x); \\ \max\{g(x)a, f(x)b\}, & \text{if } f(x) \leq g(x) \end{cases} \\
 &= \begin{cases} f(x)a, & \text{if } f(x) > g(x); \\ f(x)b(= g(x)b), & \text{if } f(x) = g(x); \\ g(x)a, & \text{if } f(x) < g(x). \end{cases}
 \end{aligned}$$

If we restrict the domains of M' , M'' to the regular language $R = \#\Sigma^+\#$, then there is an $x' \in \Sigma'^*$ such that $\widetilde{M}'(x') \neq \widetilde{M}''(x')$ if and only if there is an $x \in \Sigma^+$ such that $f(x) = g(x)$.

The complexities of the inequivalence problem for deterministic and nondeterministic generalized sequential machines are strikingly different. Griffiths [7] has shown the problem to be undecidable for nondeterministic GSMs, and Jones, Lien, and Laaser [13] have shown the problem to be NL-complete for deterministic GSMs.

(10) First, we show Range Inequivalence is recursively enumerable. The question whether or not $\widetilde{M}'(\Sigma'^*) \neq \widetilde{M}''(\Sigma''^*)$ can be restated: Is there a y such that $y \in \widetilde{M}'(\Sigma'^*) - \widetilde{M}''(\Sigma''^*)$ or $y \in \widetilde{M}''(\Sigma''^*) - \widetilde{M}'(\Sigma'^*)$? The existentially quantified predicate is decidable using the algorithm of Theorem 4.3, so Range Inequivalence is recursively enumerable. Next, we show hardness by \leq_m^L -reduction from PCP. Let f, g be homomorphisms from Σ to Δ^+ . We construct OFTs M', M'' with input alphabet $\Sigma' = \{\#\} \cup \Sigma$ and output alphabet $\Delta' = \{a, b\} \cup \Delta$ ($\Delta < a < b$) such that $\widetilde{M}'(\Sigma'^*) \neq \widetilde{M}''(\Sigma'^*)$ if and only if there is an $x \in \Sigma^+$ satisfying $f(x) = g(x)$. Let M', M'' be constructed from f, g as in Theorem 7.2(9) (see Figure 7.3). Recall that for $x \in \Sigma^+$,

$$\begin{aligned} \widetilde{M}'(\#x\#) &= \begin{cases} f(x)a, & \text{if } f(x) > g(x); \\ f(x)a(=g(x)a), & \text{if } f(x) = g(x); \\ g(x)a, & \text{if } f(x) < g(x), \end{cases} \\ \widetilde{M}''(\#x\#) &= \begin{cases} f(x)a, & \text{if } f(x) > g(x); \\ f(x)b(=g(x)b), & \text{if } f(x) = g(x); \\ g(x)a, & \text{if } f(x) < g(x). \end{cases} \end{aligned}$$

If we restrict the domains of M', M'' to the regular language $R = \#\Sigma^+\#$, then $\widetilde{M}'(\Sigma'^*) \neq \widetilde{M}''(\Sigma'^*)$ if and only if there is an $x \in \Sigma^+$ such that $f(x) = g(x)$, because $f(x) = g(x)$ implies $\widetilde{M}''(\#x\#) = f(x)b \notin \widetilde{M}'(\Sigma'^*) \subseteq (\Delta \cup \{a\})^*$. ■

Corollary 7.3. $\text{range}(\text{OFT})$ is not effectively closed under complement.

Proof. We assume $\text{range}(\text{OFT})$ is effectively closed under complement and contradict the undecidability of Range Inequivalence. Let M', M'' be OFTs with input alphabet Σ . Construct—using Theorem 5.1 and effective closure under complement—an OFT M whose range $\widetilde{M}(\Sigma^*)$ is the symmetric difference of the ranges of M', M'' . By construction,

$$\widetilde{M}(\Sigma^*) = \widetilde{M}'(\Sigma'^*) \triangle \widetilde{M}''(\Sigma'^*) = \overline{(\widetilde{M}'(\Sigma'^*) \cup \widetilde{M}''(\Sigma'^*))} \cup \overline{(\widetilde{M}'(\Sigma'^*) \cup \widetilde{M}''(\Sigma'^*))}.$$

Then $\widetilde{M}'(\Sigma'^*) \neq \widetilde{M}''(\Sigma'^*)$ if and only if the symmetric difference $\widetilde{M}(\Sigma^*) \neq \phi$. The decidability of the latter by Theorem 7.2(1) implies the decidability of Range Inequivalence, contradicting Theorem 7.2(10). ■

Corollary 7.4. $\text{range}(\text{OFT}) \subset \text{NL}$.

Proof. We have shown $\text{range}(\text{OFT}) \subseteq \text{NL}$ in Corollary 4.4, so we show $\text{range}(\text{OFT}) \neq \text{NL}$. Immerman has recently shown [11] that $\text{NSPACE}(S(n))$ is effectively closed under complement, for $S(n) \geq \log n$. In particular, NL is effectively closed under complement and must differ from $\text{range}(\text{OFT})$, in light of Corollary 7.3. ■

The relative complexities of the decision problems considered in this section are summarized in Figure 7.4. The problems between NP-complete and RE-complete are all PSPACE-hard.

Figure 7.4. Complexities of decision problems

8. 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 provided an algorithm which computes $\widetilde{M}(x)$ for a fixed OFT M using time polynomial in $|x|$ and space linear in $|x|$. Can the optimal output function \widetilde{M} be computed by a logspace transducer? Can OFTs be used to provide efficient

reductions—via the optimal output function—between decision problems? In particular, OFTs which are not bounded by a few alternations of maximization and minimization might be used to closely relate problems which are not otherwise obviously related. We confess to not understanding the full power of alternation, despite providing a polynomial time bound on computing an arbitrary optimal output function.

In §4, we showed that the range $\widetilde{M}(\Sigma^*)$ of an OFT M is in NL. How tight is this upper bound? Is $\text{range}(\text{OFT}) \subseteq \text{DSPACE}(\log n)$? Are there NL-complete languages in $\text{range}(\text{OFT})$? What subclasses of NL containing the regular languages are contained in $\text{range}(\text{OFT})$? For example, are the linear context-free languages or, perhaps more likely, the real-time one-counter languages contained in $\text{range}(\text{OFT})$?

In §6, we presented a hierarchy in NL whose classes are based on the ranges of alternation-bounded OFTs. Are the levels of the hierarchy proper? In light of the fact that we failed to produce a language which potentially distinguishes \max_j^{OFT} and \min_j^{OFT} , we ask whether or not they are distinct for any $j \geq 1$. If the ranges of \max_j -OFTs and \min_j -OFTs do not differ, are the optimal output functions distinct? In particular, can an OFT having only maximizing states be simulated by one having only minimizing states? A tool is needed which would allow one to show that a language is not in \max_j^{OFT} or \min_j^{OFT} , perhaps a pumping lemma parameterized by the number of alternations. We conjecture that such a pumping lemma can be obtained for OFTs having only one alternation.

In §7, we gave completeness results establishing the complexity of several decision problems involving OFTs. We also presented four problems which are recursively enumerable and PSPACE-hard, but did not determine their precise complexity. It remains open whether or not these problems—Range Nonuniversality, Length Inequivalence, Nonunary Range, and Range Nonregularity—are decidable.

PART II. COUNTING FINITE-STATE AUTOMATA

1. Introduction

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. The counting finite-state automaton—or counting NFA—is a finite-state analogue of the counting Turing Machine of Valiant [21].

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 [21,22] 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 [6]. 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} . The following operations on languages are defined:

union	$L_1 \cup L_2 = \{x \mid x \in L_1 \vee x \in L_2\}$
intersection	$L_1 \cap L_2 = \{x \mid x \in L_1 \wedge x \in L_2\}$
concatenation	$L_1 L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$
iteration	$L^0 = \{\epsilon\}, L^{i+1} = LL^i$
Kleene closure	$L^* = \bigcup_{i \geq 0} L^i$

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

$s(k)$ = the binary representation of k without leading zeroes,

$\#(x)$ = the number represented in binary by x .

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) [5] 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 [6], 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

$$\delta'(q_i, \sigma) = \{\delta(q_i, \sigma)\} \cup \{\delta(q_i, \sigma)^j \mid 1 \leq j \leq \#(\lambda(q_i, \sigma))\},$$

$$\delta'(q_i^k, \sigma) = \{\delta(q_i, \sigma)^j \mid 1 \leq j \leq 2^{|\lambda(q_i, \sigma)|}\}.$$

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. (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 $x \in \Sigma^*$, $\#M(x) = \#M'(x) + \#M''(x)$. Let $M = (Q' \cup Q'', \Sigma, \delta, I' \cup I'', F' \cup F'')$, where δ is defined by

$$\delta(q, \sigma) = \begin{cases} \delta'(q, \sigma), & \text{if } q \in Q'; \\ \delta''(q, \sigma), & \text{if } q \in Q''. \end{cases}$$

For $x \in \Sigma^*$,

$$\begin{aligned}
\#M(x) &= \sum_{q \in I' \cup I''} \#\delta(q, x) \\
&= \sum_{q \in I'} \#\delta'(q, x) + \sum_{q \in I''} \#\delta''(q, x) \\
&= \#M'(x) + \#M''(x).
\end{aligned}$$

(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 $x \in \Sigma^*$, $\#M(x) = \#M'(x) \cdot \#M''(x)$. Let $M = (Q' \times Q'', \Sigma, \delta, I' \times I'', F' \times F'')$, where δ is defined by

$$\delta((q', q''), \sigma) = \delta'(q', \sigma) \times \delta''(q'', \sigma).$$

We prove, by induction on $|x|$, the following claim:

$$(*) \quad \#\delta((q', q''), x) = \#\delta'(q', x) \cdot \#\delta''(q'', x).$$

For the empty string ϵ ,

$$\begin{aligned} \#\delta((q', q''), \epsilon) &= \begin{cases} 1, & \text{if } (q', q'') \in F' \times F''; \\ 0, & \text{if } (q', q'') \notin F' \times F'' \end{cases} \\ &= \begin{cases} 1, & \text{if } q' \in F' \wedge q'' \in F''; \\ 0, & \text{if } q' \notin F' \vee q'' \notin F'' \end{cases} \\ &= \#\delta'(q', \epsilon) \cdot \#\delta''(q'', \epsilon), \end{aligned}$$

and for strings σx of length at least 1,

$$\begin{aligned} \#\delta((q', q''), \sigma x) &= \sum_{(p', p'') \in \delta((q', q''), \sigma)} \#\delta((p', p''), x) \\ &= \sum_{\substack{p' \in \delta'(q', \sigma) \\ p'' \in \delta''(q'', \sigma)}} \#\delta'(p', x) \cdot \#\delta''(p'', x) \\ &= \left(\sum_{p' \in \delta'(q', \sigma)} \#\delta'(p', x) \right) \cdot \left(\sum_{p'' \in \delta''(q'', \sigma)} \#\delta''(p'', x) \right) \\ &= \#\delta'(q', \sigma x) \cdot \#\delta''(q'', \sigma x). \end{aligned}$$

Applying $(*)$, we have

$$\begin{aligned} \#M(x) &= \sum_{(q', q'') \in I' \times I''} \#\delta((q', q''), x) \\ &= \sum_{\substack{q' \in I' \\ q'' \in I''}} \#\delta'(q', x) \cdot \#\delta''(q'', x) \\ &= \left(\sum_{q' \in I'} \#\delta'(q', x) \right) \cdot \left(\sum_{q'' \in I''} \#\delta''(q'', x) \right) \\ &= \#M'(x) \cdot \#M''(x). \quad \blacksquare \end{aligned}$$

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 $Q = \{q_1, \dots, q_{2^k}, p_1, \dots, p_{2^k+1}\}$, $I = \{q_1\}$, $F = \{p_1, \dots, p_{2^k+1}\}$, and δ is defined by

$$\delta(q_i, 0) = \{q_1, \dots, q_{2^k}, p_1, \dots, p_{2^k}\},$$

$$\delta(p_i, 0) = \{p_1, \dots, p_{2^k+1}\}.$$

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 $n > 0$,

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

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 $\text{REG} \subset \text{CFL}$.

■

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.

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 $\text{NSPACE}(n)$.

Hereafter, we assume that the reader is familiar with the basic concepts of language and decidability theory, the Turing machine model, and the time-bounded and space-bounded complexity classes $\text{DTIME}(S(n))$, $\text{NTIME}(S(n))$, $\text{DSpace}(S(n))$, and $\text{NSpace}(S(n))$. We will be concerned mainly with the following complexity classes:

$$L = \text{DSPACE}(\log n),$$

$$\text{NL} = \text{NSPACE}(\log n),$$

$$P = \bigcup_{i \geq 1} \text{DTIME}(n^i),$$

$$\text{NP} = \bigcup_{i \geq 1} \text{NTIME}(n^i),$$

$$\text{CSL} = \text{NSPACE}(n),$$

$$\text{PSPACE} = \bigcup_{i \geq 1} \text{DSPACE}(n^i) = \bigcup_{i \geq 1} \text{NSPACE}(n^i).$$

For formal definitions of these notions, refer to [8].

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 $x \in \Sigma^*$. ■

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:

$$\#\delta(q_1, 0^l) = l;$$

$$\#\delta(q_2, 0^l) = 1.$$

For $l = 0$,

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

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

and for $l > 0$,

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

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

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 [20] and Stockmeyer [18]. Let M be an NFA with input alphabet Σ .

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

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 [17] 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 $\text{range}(\#\text{NFA})$ includes $\text{range}(\#\text{DFT})$ —the ranges of deterministic finite-state transducers. It follows from the Generalized Sequential Machine results of Ginsburg and Greibach [6] 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 $\text{range}(\#\text{NFA})$?

In §4, we showed that the range of a counting NFA M 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 $\text{range}(\#\text{NFA})$? 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 $\#\text{NFA}$ or a language is not in $\text{range}(\#\text{NFA})$? 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] Chandra, A.K., Kozen, D.C. and Stockmeyer, L.J. “Alternation.” *JACM* **28** (1981), 114–133.
- [2] Floyd, R.W. “New proofs and old theorems in logic and formal linguistics.” Computer Associates Inc., Wakefield, Mass., 1964.
- [3] Garey, M.R. and Johnson, D.S. “Complexity results for multiprocessor scheduling under resource constraints.” *SIAM J. Comput.* **4** (1975), 397–411.
- [4] Garey, M.R. and Johnson, D.S. “Strong NP-completeness Results: Motivation, Examples, and Implications.” *JACM* **25** (1978), 499–508.
- [5] Ginsburg, S. “Examples of abstract machines.” *IEEE Trans. on Electronic Computers* **11** (1962), 132–135.
- [6] 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.
- [7] Griffiths, T.V. “The Unsolvability of the Equivalence Problem for Λ -Free Nondeterministic Generalized Machines.” *JACM* **15** (1968), 409–413.
- [8] Hopcroft, J.E. and Ullman, J.D. *Introduction to automata theory, languages, and computation*. Addison-Wesley, Reading, Mass., 1979.
- [9] Hunt, H.B. III and Rosenkrantz, D.J. “Computational parallels between the regular and context-free languages.” *SIAM J. Comput.* **7** (1978), 99–114.
- [10] Ibarra, O.H. “The unsolvability of the equivalence problem for ϵ -free NGSM’s with unary input (output) alphabet.” *Proc. Eighteenth Annual IEEE Symposium on Foundations of Computer Science* (1977), 74–81.
- [11] Immerman, N. “Nondeterministic Space is Closed Under Complement.” Yale University Tech. Report 552, 1987.
- [12] Jones, N.D. “Space-Bounded Reducibility among Combinatorial Problems.” *JCSS* **11** (1975), 68–85.
- [13] Jones, N.D., Lien, Y.E. and Laaser, W.T. “New Problems Complete for Nondeterministic Log Space.” *Math. Systems Theory* **10** (1976), 1–17.
- [14] Krentel, M.W. “The Complexity of Optimization Problems.” *Proc. Eighteenth Annual ACM Symposium on the Theory of Computing* (1986).

- [15] Post, E. "A variant of a recursively unsolvable problem." *Bull. AMS* **52** (1946), 264–268.
- [16] Savitch, W.J. "Relationships between nondeterministic and deterministic tape complexities." *JCSS* **4** (1970), 177–192.
- [17] 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.
- [18] 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.
- [19] Stockmeyer, L.J. "The polynomial-time hierarchy." *Theor. Comput. Sci.* **3** (1976), 1–22.
- [20] 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.
- [21] Valiant, L.G. "The complexity of computing the permanent." *Theor. Comput. Sci.* **8** (1979), 189–201.
- [22] Valiant, L.G. "The complexity of enumeration and reliability problems." *SIAM J. Comput.* **8** (1979), 410–421.
- [23] Wrathall, C. "Complete sets and the polynomial-time hierarchy." *Theor. Comput. Sci.* **3** (1976), 23–33.

ACKNOWLEDGEMENTS

I would like to acknowledge the contribution to this research by Prof. Giora Slutzki, my Ph.D. advisor. His insight and knowledge of the relevant literature were directly responsible for some of these results, and positively influenced all of them. This work is the product of an enjoyable two-year collaboration from which I have benefited immeasurably.

I would also like to acknowledge the support of my wife Nancy, who shared these years and apparently enjoyed them as much as I did, and my parents Marilyn and Donald Rich, whose influence is most responsible for my accomplishment.