

The Complexity of Optimizing Finite-State Transducers

Craig A. Rich
Computer Science Department
California State Polytechnic University, Pomona
Pomona, CA 91768, USA
e-mail: carich@csupomona.edu

Giora Slutzki
Computer Science Department
Iowa State University
Ames, IA 50011, USA
e-mail: slutzki@cs.iastate.edu

Revised March 1993

Abstract

An optimizing finite-state transducer is a nondeterministic finite-state transducer in which states are either maximizing or minimizing. In a given state, the optimal output is the maximum or minimum—over all possible transitions—of the transition output concatenated with the optimal output of the resulting state. The ranges of optimizing finite-state transducers form a class in NL which includes a hierarchy based on the number of alternations of maximizing and minimizing states in a computation.

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 [2], where they are called Generalized Sequential Machines (GSM's). 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 [6], 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 OFT's and alternating finite automata. The maximizing (minimizing) states in OFT's 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 [3] 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 OFT's 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 OFT's, which we denote $\text{range}(\text{OFT})$, is in NL and contains non-context-free languages. In §5, we present a hierarchy in NL based on the ranges of OFT's whose computations have a bounded number of alternations between maximizing and minimizing states. Finally, in §6, we summarize and present open questions about optimizing finite-state transducers.

2. Preliminary Definitions

Let \mathcal{N} stand for the set of all natural numbers and let $\|S\|$ denote the cardinality of the set S . We shall write $x \leq y$ to denote that string x is lexicographically less than or equal to string y . The following operations on languages are defined:

$$\begin{array}{ll} \text{maximum} & \max L = \{x \mid x \in L \wedge \forall y \in L, y \leq x\} \\ \text{minimum} & \min L = \{x \mid x \in L \wedge \forall y \in L, x \leq y\} \\ \text{length} & |L| = \{|x| \mid x \in L\} \end{array}$$

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 . 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 (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

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

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

A simple induction on $|x|$ shows that $\|\tilde{\delta}(q, x)\| \leq 1$. 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 $\tilde{M}(L) = \bigcup_{x \in L} \tilde{M}(x)$. 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 $\text{range}(\text{OFT})$.

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 computations of an OFT on a particular input are perhaps best illustrated as branches of a *computation tree* having the initial state as root. The optimal output $\tilde{M}(x)$ 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.

3. An Algebraic Characterization

The method of computing $\tilde{M}(x)$ 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 $\tilde{M}(x)$.

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 . We note here without proof that in general, optimizing product is not associative; however, if the signatures involved are exclusively \max or \min , then the optimizing product is associative. Hereafter, we stipulate that optimizing product associates to the right.

Now we show how optimizing product can be used to compute $\tilde{M}(x)$. 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

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

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

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

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

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

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

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

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 $|\tilde{\delta}(q_i, \sigma_n \dots \sigma_1)|$ 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 $\tilde{\delta}(q, x)$ 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.

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 $v_1 = \tilde{\delta}(q_1, \sigma_n \dots \sigma_1) = \widetilde{M}(x)$:

```

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 is a natural number 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 languages and representations of languages 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, which we state here without proof, see [7].

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) $\text{rep}_y(w_1 w_2) = \text{rep}_y(w_1, \text{rep}_y(w_2))$.

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

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

$A *^y B$ 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}^σ 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. $\text{range}(\text{OFT}) \subseteq \text{NL}$.

Other properties of $\text{range}(\text{OFT})$ that we have shown, but not included here, are (i) for every regular language R and OFT M , both R and $\widetilde{M}(R)$ are in $\text{range}(\text{OFT})$, (ii) $\text{range}(\text{OFT})$ contains non-context-free languages, and (iii) $\text{range}(\text{OFT})$ is effectively closed under union, concatenation, and Kleene closure, see [7]. We have also investigated the complexity of many decision problems dealing with $\text{range}(\text{OFT})$. For example, we prove that the range inequivalence problem for OFTs is undecidable, and conclude that $\text{range}(\text{OFT})$ is not effectively closed under complement and is, therefore, properly included in NL, see [5]. We present other examples of problems about $\text{range}(\text{OFT})$ which are NL-complete, NP-complete, PSPACE-complete, and RE-complete, see [7].

5. A Hierarchy in NL

We have shown that the range of an arbitrary OFT is in NL; therefore, the classes of ranges of \max_j - and \min_j -OFT's 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 relevant closure property—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 5.1.

- (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 , 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 $R \in \max_1^{\text{OFT}} \cap \min_1^{\text{OFT}}$. We have a non-regular (in fact, non-context-free) language in \max_1^{OFT} , and a non-context-free language in \min_1^{OFT} can be similarly obtained, see [7].

(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' be a \max_j -OFT with input alphabet Σ' and construct a \min_{j+1} -OFT M with input alphabet $\Sigma = \{\#\} \cup \Sigma'$ such that $\widetilde{M}(\Sigma^*) = \widetilde{M}'(\Sigma'^*)$. Add a new minimizing start state q_1 to M' such that $\delta(q_1, \#) = \{(q'_1, \varepsilon)\}$. M is a \min_{j+1} -OFT, since M' is a \max_j -OFT. If we restrict the domain of M to the regular language $R = \#\Sigma'^*$, then $\widetilde{M}(\Sigma^*) = \widetilde{M}'(\Sigma'^*)$. The inclusion $\min_j^{\text{OFT}} \subseteq \max_{j+1}^{\text{OFT}}$ 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 5.1 are depicted in Figure 5.1.

After a technical lemma whose proof is omitted (but given in [7]), 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 $\text{minmax}_1(l_0, l_1) = \min(l_0, l_1)$.

NL

OFTH

$$\begin{array}{cc} \max_3^{\text{OFT}} & \min_3^{\text{OFT}} \\ \max_2^{\text{OFT}} & \min_2^{\text{OFT}} \\ \max_1^{\text{OFT}} & \min_1^{\text{OFT}} \end{array}$$

proper

REG

Figure 5.1. A hierarchy in NL

Lemma 5.2. For $j \geq 1$, there is a \max_j - and \min_j -OFT M such that

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

and there is a \min_j -(\max_j -)OFT M such that

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

Theorem 5.3. For $j \geq 1$,

$$\begin{aligned} L_1 &= \{0^k \#0^{l_0} \dots \#0^{l_j} \mid k \geq \max \min_j(l_0, \dots, l_j)\} \in \max_j^{\text{OFT}} \cap \min_j^{\text{OFT}}, \\ L_2 &= \{0^k \#0^{l_0} \dots \#0^{l_j} \mid k = \max \min_j(l_0, \dots, l_j)\} \in \max_{j+1}^{\text{OFT}} \cap \min_{j+1}^{\text{OFT}}. \end{aligned}$$

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 5.2, 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

$$\begin{aligned} \widetilde{M}(x_1 x_2) &= \max\{x_1 x_2, \widetilde{M}'(x_2)\} \\ &= \begin{cases} x_1 x_2, & \text{if } |x_1 x_2| \geq |\widetilde{M}'(x_2)|; \\ \widetilde{M}'(x_2), & \text{otherwise} \end{cases} \\ &= \begin{cases} x_1 x_2, & \text{if } k + j + 1 + \Sigma l_i \geq j + 1 + \Sigma l_i + \max \min_j(l_0, \dots, l_j); \\ \text{a string of a's,} & \text{otherwise} \end{cases} \\ &= \begin{cases} x_1 x_2, & \text{if } k \geq \max \min_j(l_0, \dots, l_j); \\ \text{a string of a's,} & \text{otherwise.} \end{cases} \end{aligned}$$

($L_1 \in \min_j^{\text{OFT}}$) 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 5.2,

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

$$\begin{aligned} \widetilde{M}(x_1 x_2) &= \min\{x_1 x_2, b^{2k} \widetilde{M}'(x_2)\} \\ &= \begin{cases} x_1 x_2, & \text{if } |x_1 x_2| \leq |b^{2k} \widetilde{M}'(x_2)|; \\ b^{2k} \widetilde{M}'(x_2), & \text{otherwise} \end{cases} \\ &= \begin{cases} x_1 x_2, & \text{if } k+j+1+\Sigma l_i \leq 2k+j+1+\Sigma l_i - \max \min_j(l_0, \dots, l_j); \\ \text{a string of b's,} & \text{otherwise} \end{cases} \\ &= \begin{cases} x_1 x_2, & \text{if } k \geq \max \min_j(l_0, \dots, l_j); \\ \text{a string of b's,} & \text{otherwise.} \end{cases} \end{aligned}$$

($L_2 \in \max_{j+1}^{\text{OFT}}$) 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 5.2, 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 5.2, 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

$$\begin{aligned} \widetilde{M}(x_1 x_2) &= \max\{x_1 x_2, \widetilde{M}'(x_2), a^{2k} \widetilde{M}''(x_2)\} \\ &= \begin{cases} x_1 x_2, & \text{if } (|x_1 x_2| \geq |\widetilde{M}'(x_2)|) \wedge (|x_1 x_2| \geq |a^{2k} \widetilde{M}''(x_2)|); \\ \text{a string of a's,} & \text{otherwise} \end{cases} \\ &= \begin{cases} x_1 x_2, & \text{if } (k+j+1+\Sigma l_i \geq j+1+\Sigma l_i + \max \min_j(l_0, \dots, l_j)) \wedge \\ & (k+j+1+\Sigma l_i \geq 2k+j+1+\Sigma l_i - \max \min_j(l_0, \dots, l_j)); \\ \text{a string of a's,} & \text{otherwise} \end{cases} \\ &= \begin{cases} x_1 x_2, & \text{if } k \geq \max \min_j(l_0, \dots, l_j) \wedge k \leq \max \min_j(l_0, \dots, l_j); \\ \text{a string of a's,} & \text{otherwise} \end{cases} \\ &= \begin{cases} x_1 x_2, & \text{if } k = \max \min_j(l_0, \dots, l_j); \\ \text{a string of a's,} & \text{otherwise.} \end{cases} \end{aligned}$$

($L_2 \in \min_{j+1}^{\text{OFT}}$) 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 5.2, 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 5.2, 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

$$\begin{aligned} \widetilde{M}(x_1 x_2) &= \min\{x_1 x_2, \widetilde{M}'(x_2), b^{2k} \widetilde{M}''(x_2)\} \\ &= \begin{cases} x_1 x_2, & \text{if } (|x_1 x_2| \leq |\widetilde{M}'(x_2)|) \wedge (|x_1 x_2| \leq |b^{2k} \widetilde{M}''(x_2)|); \\ \text{a string of b's,} & \text{otherwise} \end{cases} \\ &= \begin{cases} x_1 x_2, & \text{if } (k+j+1+\Sigma l_i \leq j+1+\Sigma l_i + \max \min_j(l_0, \dots, l_j)) \wedge \\ & (k+j+1+\Sigma l_i \leq 2k+j+1+\Sigma l_i - \max \min_j(l_0, \dots, l_j)); \\ \text{a string of b's,} & \text{otherwise} \end{cases} \\ &= \begin{cases} x_1 x_2, & \text{if } k \leq \max \min_j(l_0, \dots, l_j) \wedge k \geq \max \min_j(l_0, \dots, l_j); \\ \text{a string of b's,} & \text{otherwise} \end{cases} \\ &= \begin{cases} x_1 x_2, & \text{if } k = \max \min_j(l_0, \dots, l_j); \\ \text{a string of b's,} & \text{otherwise.} \quad \blacksquare \end{cases} \end{aligned}$$

There are several open questions in conjunction with the hierarchy that we have presented. Which inclusions shown in Figure 5.1 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.

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 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 OFT's be used to provide efficient reductions—via the optimal output function—between decision problems? In particular, OFT's 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 §5, we presented a hierarchy in NL whose classes are based on the ranges of alternation-bounded OFT's. 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^{OFT} 's and \min_j^{OFT} 's 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 OFT's having only one alternation.

References

- [1] Chandra, A.K., Kozen, D.C., and Stockmeyer, L.J. "Alternation," *JACM* **28**: 1 (1981), 114–133.
- [2] Ginsburg, S. "Examples of abstract machines," *IEEE Trans. on Electronic Computers* **11**: 2 (1962), 132–135.
- [3] 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.
- [4] Hopcroft, J.E., and Ullman, J.D. "Introduction to automata theory, languages, and computation," Addison-Wesley, Reading, Mass., 1979.
- [5] Immerman, N. "Nondeterministic Space is Closed Under Complement," Yale University Technical Report, 1987.
- [6] Krentel, M.W. "The Complexity of Optimization Problems," *Proc. Eighteenth Annual ACM Symposium on the Theory of Computing* (1986).
- [7] Rich, C.A. "The Complexity of Two Finite-State Models—Optimizing Transducers and Counting Automata," Ph.D. Dissertation, Iowa State University, 1988.