CHAPTER IV

Classes of Invertible Functions

In this chapter, three classes of machines are presented which compute only functions with inverses computable in polynomial time. Essentially any other natural class of machines which is not a subclass of one of these three classes will contain a machine computing a function, no inverse for which is computable in polynomial time.

We also investigate classes of circuits which compute only functions with easy inverses. Among other results, we show that functions computed by narrow circuits have inverses computed by shallow circuits.

Classes of Machines Computing Invertible Functions

Let $\mathbf{C}_1$ be the class of all Turing machines M such that for some nonnegative integers j and l, M has

(a) j one-way read-only input tapes

(b) one logspace-bounded work tape

(c) one pushdown store

(d) l one-way write-only output tapes.

Let $\mathbf{C}_2$ be the class of all Turing machines M such that for some nonnegative integers j and l, M has

(a) j one-way read-only input tapes

(b) one finite-crossing read-write input tape

(c) one logspace-bounded work tape

(d) l one-way write-only output tapes.

Let $\mathbf{C}_3$ be the class of all Turing machines M such that for some nonnegative integers j, k, l and m, M has

(a) j one-way read-only input tapes

(b) one unrestricted two-way input tape

(c) k counters, each of which makes at most m reversals

(d) l one-way write-only output tapes.

We will show that any honest function computed by a machine in $\mathbf{C}_1$, $\mathbf{C}_2$, or $\mathbf{C}_3$ has an easy inverse. To that end, we first prove a preliminary theorem.

Theorem 4.1:

(a) Let f be an honest function computed by a machine in $\mathbf{C}_1$. Then range(f) $\in$ PUNC.

(b) Let f be an honest function computed by a machine in $\mathbf{C}_2$. Then range(f) $\in$ NLOG.

(c) Let f be an honest function computed by a machine in $\mathbf{C}_3$. Then range(f) $\in$ NLOG.

Proof: We will prove (a), (b) and (c) only for the case in which there is one input tape. It is straightforward to extend the proofs to handle the case where there are additional one-way input tapes.

(a) Let f be an honest function computed by a machine M in $\mathbf{C}_1$. Assume without loss of generality that on each move M either pushes or pops symbols from the stack. Assertion (a) could be easily proved by constructing a one-way NAuxPDA accepting range(f). However, in order to facilitate proofs of subsequent theorems, we will prove (a) by presenting a logspace-bounded alternating Turing machine M′ which uses treesize $n^{O(1)}$ and accepts a set L such that for a certain function h (given below) which is computable in polynomial time, $y\#h(|y|) \in$ L iff $y \in$ range(f). L is thus in NC, and hence range(f) $\in$ PUNC.

We will use a definition of *surface configuration* which is slightly different from that used in the proof of Theorem 3.3; in this chapter we wish to record the positions of the input and output heads as well. A *surface configuration* of a machine M in $\mathbf{C}_1$ on input x is a 6-tuple (q,w,i,j,l,Γ,a), where q is a state of M, w is a string of worktape symbols (the *worktape contents*), i is an integer, $1 \leq i \leq |x|$ (the *input head position*), j is an integer, $1 \leq j \leq |f(x)|$ (the *output head position*), l is an integer, $1 \leq l \leq |w|$ (the *worktape head position*), Γ is a pushdown symbol (the *stack top*), and a is the i-th symbol of the input x. A pair of surface configurations (C,D) is a *realizable pair* if M can start a computation with surface configuration C with some stack height h, and eventually enter a state with surface configuration D with stack height h, with the stack height never less than h during the computation from C to D. Since M pushes or pops a symbol on every move, it is easy to verify that (C,D) is a realizable pair iff one of the following holds:

(i) C = D,

(ii) there is a configuration E such that (C,E) and (E,D) are realizable pairs, or

(iii) there is a realizable pair (E,F) such that C $\vdash$ E via a push of Γ and F $\vdash$ D via a pop of Γ, for some stack symbol Γ.

(These definitions are similar to those used in e.g. [Ru-81].)

Let p be a polynomial such that, if $y \in$ range(f), then there is an x such that $|x| \leq p(|y|)$. Such a p exists, since f is honest. Let W(n) be the set of all possible surface configurations of M on inputs of length $\leq p(n)$. Let $h(n)$ be a listing of all realizable pairs (C,D) such that C and D are both in W(n), the input head position in C equals the input head position in D, and the output head position in C equals the output head position in D. Thus (C,D) appears in $h(n)$ iff there is a computation from C to D which leaves the input and output heads fixed. It is clear that $h(n)$ is computable in polynomial time.

Now consider the alternating Turing machine M′ which on input $y\#h$ performs the following computation.

Guess a binary number n and check universally that $n \leq p(|y|)$. (n is the guessed length of x.)
C := initial surface configuration for inputs of size n
D := final surface configuration for inputs of size n and outputs of length $|y|$
while C $\neq$ D and (C,D) does not appear in h
existentially choose (1) or (2) below:
(1) (This corresponds to checking if condition (iii) holds.)
Guess a pair of configurations (E,F) such that C $\vdash$ E via a push of Γ and F $\vdash$ D via a pop of Γ, for some stack symbol Γ.
If the move C $\vdash$ E produces the j-th output symbol, check that the symbol produced matches the j-th symbol of y.
If the move F $\vdash$ D produces the j-th output symbol, check that the symbol produced matches the j-th symbol of y.
C := E
D := F
(2) (This corresponds to checking if condition (ii) holds.)
Guess a configuration E, and universally choose (2.1) and (2.2) below:
(2.1) D := E
(2.2) C := E
endwhile
At this point, C = D, or (C,D) appears in h. Halt and accept.

It is straightforward to verify that M′ accepts an input $y\#h(|y|)$ iff there is a string x such that M outputs y on input x. M′ simulates the computation of M by verifying that the initial configuration C and the final configuration D of M constitute a realizable pair, and that there is a computation starting in C and ending in D which outputs y. The string x is determined by the sequence of surface configurations which witness that (C,D) is a realizable pair.

The size of any accepting computation tree of M′ on input $y\#h(|y|)$ is bounded by a polynomial in the number of times which M moves its input and output heads on input y. The number of times which M moves its input and output heads is in turn bounded by a polynomial in $|y|$. M′ can be modified to accept in treesize $n^{O(1)}$ on *all* inputs (not just those of the form $y\#h(|y|)$). M′ then accepts a language in NC.

(b) Let f be an honest function computed by a machine M in $\mathbf{C}_2$. In order to simplify the presentation of the proof, we will only prove (b) only for the case in which M's input head

changes direction only k times, where k is even, and furthermore, each of these changes in direction takes place after making a complete sweep across the input tape. A computation of M can thus be divided into $k+1$ phases, with phase i occurring after the i-th time the input head changes direction. It is straightforward to extend the proof presented here to prove the more general assertion (b).

We now construct a nondeterministic logspace-bounded Turing machine M′ which accepts range(f).At specified points j in the computation of M′, the worktape of M′ will contain k configurations of M, $C_0(j)$, $C_1(j)$,..., $C_k(j)$. Each $C_i(j)$ is a configuration of M which occurs during phase i of M's computation, and the input head position is the same in each configuration $C_i(j)$. This is diagrammed in Figure 4-1. An accepting computation of M' will be viewed as representing a computation of M in the way suggested by Figure 4-2. (The thin arrow represents the path of M's computation.)

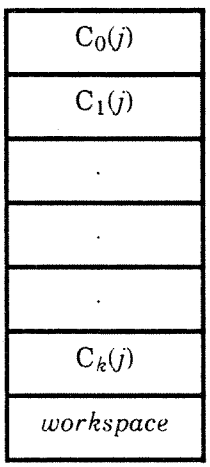

Figure 4-1: a configuration of M′

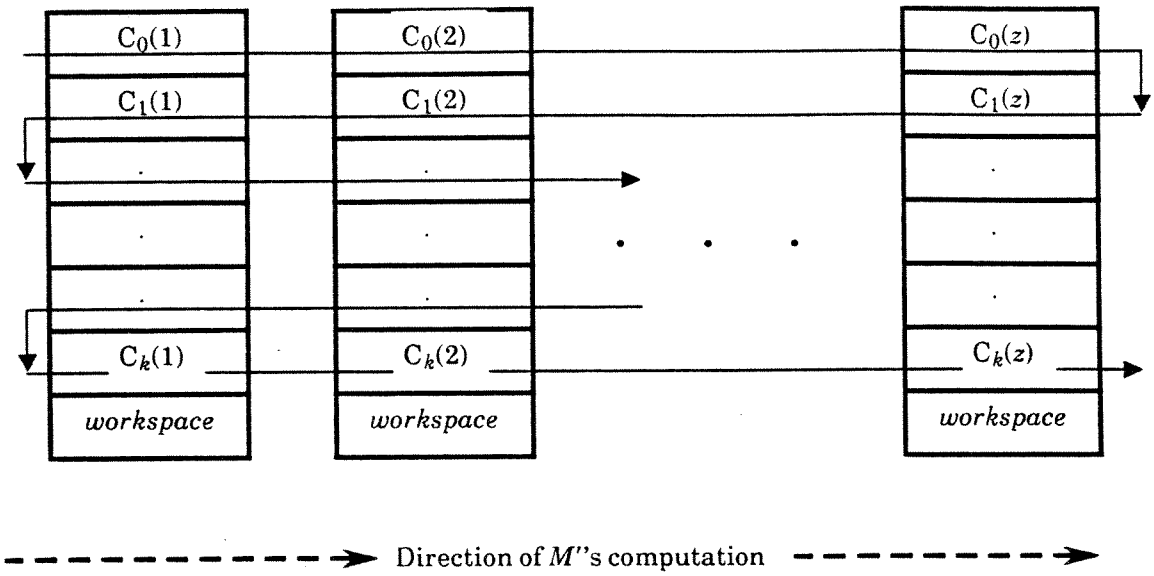

Figure 4-2: a representation of an accepting computation of M'

In these schematic diagrams, $C_i(z) = C_{i+1}(z)$, and $C_i(j) \vdash C_i(j+1)$ or $C_i(j) = C_i(j+1)$ for all $j < z$, for all even $i \leq k$, and $C_i(1) = C_{i+1}(1)$, and $C_i(j) \vdash C_i(j-1)$ or $C_i(j) = C_i(j-1)$ for all $j > 1$, for all odd $i \leq k-1$.

On input y, M' operates by first marking off log $|y|$ on its worktape. M' then guesses a configuration $(C_0(1), C_1(1),..., C_k(1))$, where $C_0(1)$ is an initial configuration, and for all i $C_i(1)$ is a configuration in which the head on the input tape of M is scanning the left endmarker. In addition M' checks that $C_i(1) = C_{i+1}(1)$ for all odd i.

If M' is in configuration $(C_0(j), C_1(j),..., C_k(j))$, then M' does the following tasks.

1) For $i := 0$ to k
   if $C_i(j)$ records that the output symbol at position r is "a,"
   then check that the r-th symbol of y is "a."

2) If $C_i(j) = C_{i+1}(j)$ for all even i, and $C_1(j)$ is a configuration in which M is scanning the right endmarker,
   then check that $C_k(j)$ is a halting configuration where the number of output symbols written in $C_k(j)$ is equal to $|y|$; if so, halt and accept.
3) Guess a configuration $C_i(j+1)$ for each i, $0 \leq i \leq k$, such that
   a) for some i, $C_i(j) \neq C_i(j+1)$
   b) M is scanning the same symbol in the same position on tape 1 in each of the $C_i(j+1)$
   c) For all even i, $C_i(j) \vdash C_i(j+1)$ or $C_i(j) = C_i(j+1)$
      For all odd i, $C_i(j+1) \vdash C_i(j)$ or $C_i(j+1) = C_i(j)$.
4) Enter configuration $(C_0(j+1), C_1(j+1),..., C_k(j+1))$.

It can easily be shown that M' runs in logspace, and accepts range(f).

(c) We will use the following fact about machines in C_3:

Lemma 4.2: [GI-81]

If M is a machine in C_3, then there is a machine M' in C_3 which computes the same function as M, always halts, and runs in polynomial time. Furthermore, no counter of M' makes more than one reversal during any computation.

Let f be an honest function computed by a machine M in C_3 which satisfies the conditions of Lemma 4.2. Let us represent a configuration of M by a tuple $(q,i,j,a,b,c_1,c_2, \ldots , c_k)$, where q is the state of M, i and j are the positions of the input and output heads, respectively, a is the i-th input symbol, b is the j-th output symbol, and the contents of the k counters are given by c_1 through c_k. Since M runs in polynomial time, the counters are of polynomial length, and thus a configuration of M can be written in logspace.

Consider the computation of M on some input x. If, for some configuration $(q,i,j,a,b,c_1,c_2, \ldots ,c_k)$, $(q,i,j,a,b,c_1,c_2, \ldots ,c_k) \vdash^* (q,i,j + \delta_0,a,b',c_1 + \delta_1,c_2 + \delta_2, \ldots ,c_k + \delta_k)$ for some integers δ_0 through δ_k, then M will cycle for a while, and it follows that if l is an integer such that $c_r + l\delta_r \geq 0$ for all r, $1 \leq r \leq k$, then $(q,i,j,a,b,c_1,c_2, \ldots ,c_k) \vdash^* (q,i,j + \delta_0,a,b',c_1 + \delta_1,c_2 + \delta_2, \ldots ,c_k + \delta_k) \vdash^* \ldots \vdash^* (q,i,j + l\delta_0,a,b',c_1 + l\delta_1,c_2 + l\delta_2, \ldots ,c_k + l\delta_k)$. Thus the cycle stops only when one of the counters is emptied. Since there are k counters, and each of them

only makes one turn, and thus each of them is emptied at most once, there are at most k such cycles in any computation of M.

Let r be the number of states of M. We now construct a nondeterministic logspace-bounded machine M′ which accepts range(f). M′ will have $2kr + r$ tracks on its worktape: r tracks to model each cycle which may occur during the computation, and r tracks for each part of the computation which precedes or follows a cycle.

That is, a configuration of M′ contains $2k + 1$ bands, laid out as in Figure 4-3. The

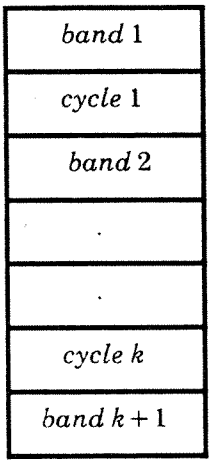

Figure 4-3: a configuration of M′

bands labelled "*band i*" correspond to parts of the computation which are not part of any cycle.

Note that if the i-th input cell of M is visited more than r times, M must be in the same state during at least two of those visits. Thus we must have $(q,i,j,a,b,c_1,c_2, \ldots ,c_k) \vdash^* (q,i,j + \delta_0,a,b',c_1 + \delta_1,c_2 + \delta_2, \ldots ,c_k + \delta_k)$ for some state q and for some integers δ_0 through δ_k, and hence we have a cycle. Therefore, parts of the computation which are not part of a cycle are r-crossing bounded. These r-crossing bounded parts of the computation can be simulated

exactly as in part (b). Similarly, a single pass through a cycle is an r-crossing bounded computation, and can be simulated in the same way.

M′ works very similarly to the machine which was defined in part (b). To check that the computations recorded in *band s* and *cycle s* are consistent with each other, it is sufficient that at some point, M′ is in a configuration in which the final configuration of M in *band s* matches the initial configuration of M in *cycle s*. To check that the computations recorded in *cycle s* and *band s* + 1 are consistent with each other requires only the following computation:

> Let the cycle recorded in *cycle s* be
> $(q,i,j,a,b,c_1,c_2,\ldots,c_k) \vdash^* (q,i,j+\delta_0,a,b',c_1+\delta_1,c_2+\delta_2,\ldots,c_k+\delta_k)$.
> Let l be the largest integer such that $c_r + l\delta_r \geq 0$ for all r, $1 \leq r \leq k$.
> Check that symbols $j+1$ through $j+\delta_0$ of y are repeated l consecutive times in y.
> Check that the initial configuration of M in *bands* + 1 is $(q,i,j+l\delta_0,a,b',c_1+l\delta_1,c_2+l\delta_2,\ldots,c_k+l\delta_k)$

Although the details of a complete proof are tedious, it is nonetheless straightforward to show that such a machine M′ can be constructed which runs in logspace and accepts range(f). □

We remark that the proof of Theorem 4.1 does not rely on the fact that the automaton computing the function f is deterministic. In particular, if M is a nondeterministic machine in $\mathbf{C}_1$ or $\mathbf{C}_2$ then the set $\{y \mid$ for some x with $|x| \leq p(|y|)$, there is a computation of M on input x which outputs $y\}$ is in PUNC (NLOG).

Theorem 4.3:

(a) Let f be an honest function computed by a machine in $\mathbf{C}_1$. Then f has an inverse in PUNC.

(b) Let f be an honest function computed by a machine in $\mathbf{C}_2$. Then f has an inverse in NC.

(c) Let f be an honest function computed by a machine in $\mathbf{C}_3$. Then f has an inverse in NC.

Proof:

(b) and (c) Let f be an honest function computed by a machine M in C_2 or C_3. Let M′ be the machine constructed in the proof of Theorem 4.1. The set L′ = {C#w | M′, when started in configuration C with input w, accepts} is in NC. Thus by Proposition 1.6 there is a logspace-bounded alternating Turing machine M_1 which accepts the *complement* of L′, such that every accepting computation tree of M_1 has at most $2^{\log^{O(1)}n}$ nodes.

Consider the alternating Turing machine M_2 which, on input w, simulates M′ in the following manner. If M′ makes an existential move choosing between configurations C and D, then M_2 performs the following steps:

Existentially choose (1) or (2)
(1) Simulate M′ directly from configuration C
(2) Universally choose (2.1) and (2.2)
(2.1) Simulate M′ directly from configuration D
(2.2) Simulate M_1 on input C#w, and thus accept iff M′ rejects w when started in configuration C with input w.

M_2 accepts range(f), since M′ does. If M_2 accepts on input y, then there is *exactly one* accepting computation tree of M_2 for y.

An accepting tree of M_2 on input y consists of an accepting computation of M′, where potentially every node C in that computation has attached to it an accepting tree of M_1 on input C#y. Thus M_2 accepts in treesize $n^{O(1)}2^{\log^{O(1)}n} = 2^{\log^{O(1)}n}$.

Recall that each accepting computation tree of M′, and hence of M_2, determines a string x such that $f(x) = y$. Thus for each y in range(f), M_2 selects a unique x such that $f(x) = y$. Let $g(y)$ denote this x. Clearly, g is an inverse of f.

Now consider the machine M_3 which on input i#y simulates M_2 on input y, but rejects whenever it stores a configuration of M in which the i-th input symbol is 0. M_3 accepts iff the i-th bit of $g(y)$ is 1. By Proposition 2.1, g is in NC.

(a) Let f be an honest function computed by a machine M in C_1. Let M′ be the machine constructed in the proof of Theorem 4.1. The set L′ = {C#w | M′, when started in

configuration C with input w, accepts} is in NC. Thus by Proposition 1.6 there is a logspace-bounded alternating Turing machine M_1 which accepts the *complement* of L, such that every accepting computation tree of M_1 has at most $2^{\log^{O(1)}n}$ nodes.

Consider the alternating Turing machine M_2 which, on input w, simulates M′ in the following manner. M_2 simulates all universal moves of M′ directly, and if M′ makes an existential move choosing between configurations C and D, then M_2 performs the following steps:

Existentially choose (1) or (2)
(1) Simulate M′ directly from configuration C
(2) Universally choose (2.1) and (2.2)
(2.1) Simulate M′ directly from configuration D
(2.2) Simulate M_1 on input C#w, and thus accept iff M′ rejects w when started in configuration C with input w.

M_2 accepts $y\#h(|y|)$ iff $y \in$ range(f), just as M′ does. If M_2 accepts an input $y\#h(|y|)$, then there is *exactly one* accepting computation tree of M_2 for w.

An accepting tree of M_2 on input $y\#h(|y|)$ consists of an accepting computation tree of M′, where potentially every node C in that computation tree has attached to it an accepting tree of M_1 on input C$\#y\#h(|y|)$. Thus M_2 accepts in treesize $n^{O(1)}2^{\log^{O(1)}n} = 2^{\log^{O(1)}n}$.

Recall that each accepting computation tree of M′, and hence of M_2, determines a string x such that $f(x) = y$. Thus for each y in range(f), M_2 selects a unique x such that $f(x) = y$. Let $g(y)$ denote this x. Clearly, g is an inverse of f.

Now consider the machine M_3 which on input $i\#y$ simulates M_2 on input $y\#h(|y|)$, but rejects whenever it stores a configuration of M in which the i-th input symbol is 0. M_3 accepts iff the i-th bit of $g(y)$ is 1. M_3 is able to construct the bits of $h(|y|)$ as it needs them, without accessing the input. The total number of nodes which access the input in an accepting tree of M_3 is therefore the same as in an accepting tree of M_2, and M_2 accepts in treesize $2^{\log^{O(1)}n}$. Thus g is in PUNC. □

Corollary 4.4:

Every honest function which is computed in time $2^{\log^{O(1)} n}$ on a machine in $\mathbf{C}_1$ has an inverse in NC.

Proof: Immediate from the proof of Theorem 4.3. □

Class $\mathbf{C}_3$ contains the class of all two-way DFA transducers, which have often been studied (see, e.g., [Gr-78a, Gr-78b, Ib-82]). Class $\mathbf{C}_2$ contains the class of one-way logspace machines, which were considered in [HIM-78,.HM-81]. Class $\mathbf{C}_1$ contains the class of all DPDA transducers, which were studied in, e.g., [AU-72, VS-78, Ib-83]. It had not been observed before that the functions computed by these classes of machines have polynomial-time-computable inverses.

Corollary 4.5:

Every honest two-way DFA transduction has an inverse in NC.

Every honest 1-L reduction has an inverse in NC.

Every honest DPDA transduction has an inverse in NC.

The techniques used above can also be used to show that other classes of functions have inverses which can be computed relatively quickly. Note that *any* honest polynomial-time function has an inverse which can be computed in time $2^{n^{O(1)}}$, and it is not known if any better upper bound on the complexity of inverting functions exists. Thus any function which has an inverse computable in subexponential time has an inverse which may be considered to be "easy" in a sense. The existence of a subexponential-time algorithm for computing the inverse of a function can have practical significance [Ad-79].

Corollary 4.6:

Let $\mathbf{C}_1(B)$ be the class of all one-way $B(n)$-space bounded AuxPDA's. where $\log S(n) = o(\log n)$.

Let $\mathbf{C}_2(B)$ be the class of all $S(n)$-space bounded Turing machines whose input heads are $C(n)$-crossing bounded, where $S(n)C(n) = B(n)$.

Let $\mathbf{C}_3(B)$ be the class of all multicounter machines where each counter is $B(n)$-reversal bounded.

If f is a function computed by a machine in $\mathbf{C}_1(B)$, $\mathbf{C}_2(B)$, or $\mathbf{C}_3(B)$, and $\log B(n) = o(\log n)$, then range(f) $\in$ NTIME,SPACE($n^{O(1)}$,$2^{O(B(n))}$) and f has an inverse which is computable in time $2^{O(B(n))}$. Thus, f has an inverse which is computable in $o(2^{n^e})$ time for all $e > 0$.

Proof: Straightforward generalization of the proof of Theorems 4.1 and 4.3. □

As an application of Corollary 4.6, consider the class of functions which are computable in less than logspace, with no restrictions placed on the way the input head accesses the input. Although we are not able to conclude that the inverses of such functions are computable in polynomial time, we can compute an inverse in subexponential time.

Corollary 4.7:

Let f be an honest function computed in space $o(\log(n))$. Then f has an inverse computable in $o(2^{n^e})$ time for all $e > 0$.

Proof: Any Turing machine which runs in space $S(n) = o(\log n)$ has an input head which is $2^{S(n)}$-crossing bounded. Since $\log(S(n)2^{S(n)}) = S(n) + \log S(n) = o(\log n)$, the result follows from Corollary 4.6. □

Turing machines which use less than logspace have very severely-limited computing resources. Nonetheless, it is most likely the case that there are one-way functions computable by loglogspace-bounded Turing machines. (Recall that any Turing machine which uses unbounded space uses at least loglogspace infinitely often [SHL-65].)

NTIME,SPACE($n^{O(1)}$, $\log^{O(1)}n$) is the nondeterministic analog of SC; some results about this class may be found in [MS-81]. It is unlikely that NTIME,SPACE($n^{O(1)}$, $\log^{O(1)}n$) $\subseteq$ P.

Theorem 4.8:

NTIME,SPACE($n^{O(1)}$, $\log^{O(1)}n$) $\subseteq$ P $\Leftrightarrow$

every honest function computable in loglogspace has an inverse which is computable in polynomial time.

Proof: ($\Rightarrow$) Let f be an honest function computed in loglogspace. Then L = $\{x\#y \mid$ for some w, $|xw| \leq q(|y|)$ and $f(xw) = y\}$ is in NTIME,SPACE($n^{O(1)}$, $\log^{O(1)}n$).

($\Leftarrow$) Let L be accepted by a Turing machine M which uses $\log^k n$ space and runs in real time. Let v be some element of L. Then consider the function $f(x) = y$ if $x = 1\#2\# \ldots n\#C_1y_1C_2y_2 \ldots C_ny_nC_{n+1}$ where $C_i \vdash C_{i+1}$ via a move which consumes input symbol y_i for $1 \leq i \leq n$, C_{n+1} is an accepting configuration and $y_1y_2 \ldots y_n = y$ (the configurations C_i consist only of worktape contents, the input head position is not recorded); $f(x) = v$ otherwise. It is easy to show that f is an honest function computable in loglogspace, and range(f) = L. If L has an inverse computable in polynomial time, then L is in P. It is known that every language in NTIME,SPACE($n^{O(1)}$, $\log^{O(1)}n$) is logspace-reducible to some language L which is accepted by a Turing machine M which uses $\log^k n$ space and runs in real time [Su-83]. Thus if every such L were in P, NTIME,SPACE($n^{O(1)}$, $\log^{O(1)}n$) $\subseteq$ P would follow. □

Other Classes of Machines

Might it be possible to improve Theorem 4.3 by considering different classes of machines? For example, instead of the LIFO pushdown store in C_1, might one not consider a FIFO data structure, as in [BGW-79, Br-80]? Also, classes C_2 and C_3 allow one two-way input tape and one one-way input tape; might one not consider having two two-way input tapes? In fact, amid the bewildering variety of data structures and types of automata which have been

considered in the literature, must there not surely exist some class of machines which compute only invertible functions which is not covered by Theorem 4.3?

We argue that the following result shows that any significant extension of Theorem 4.3 to other classes of machines is unlikely.

Theorem 4.9:

Consider the following classes of automata:

$\mathbf{D_1}$ = the class of all two-tape two-way DFA's, where each input head makes at most one reversal.

$\mathbf{D_2}$ = the class of all two-head one-way DFA's which run in real time.

$\mathbf{D_3}$ = the class of all two-way one-counter automata.

$\mathbf{D_4}$ = the class of all two-way pushdown automata where the input head and the pushdown store each make at most one reversal.

$\mathbf{D_5}$ = the class of all real-time checking stack automata [Gr-69].

$\mathbf{D_6}$ = the class of all real-time Turing machines with two pushdown stores, where each pushdown store makes at most one reversal.

$\mathbf{D_7}$ = the class of all real-time Turing machines with two reset tapes [BGW-79] where each reset tape is reset at most once.

$\mathbf{D_8}$ = the class of all real-time Turing machines with one Post tape [Br-80].

In each class $\mathbf{D_i}$, there is a machine computing an honest function whose range is NP-complete, and which thus has no inverse computable in polynomial time unless P = NP.

Proof: We prove the result only for the classes $\mathbf{D_1}$ and $\mathbf{D_5}$. The proofs for the other classes are quite similar.

Let M be any nondeterministic Turing machine which runs in polynomial time (and which copies its input onto its worktape, so that a configuration of M contains a copy of the input). A two-tape two-way DFA can scan its tapes one way to see if tape one contains a list of

configurations $C_1C_3 \ldots C_{2m-1}$, and tape two contains a list of configurations $C_2C_4 \ldots C_{2m}$ such that $C_{2i-1} \vdash C_{2i}$ for $1 \leq i \leq m$. By scanning the tapes in the reverse direction it can check if $C_{2i} \vdash C_{2i+1}$. It is now easy to construct an honest function which can be computed by a two-way DFA whose range is the set accepted by M.

Let us now consider the stack automaton which processes input of the form $v\#x$ where x is an instance of SAT such that variables are encoded in unary and v is a binary vector. (The set of all satisfiable Boolean formulas in CNF with the variables encoded in unary is an NP-complete set [Ro-73]. A checking stack automaton can store v in its stack, and start processing x in a left-to-right manner, printing each symbol of x on its output tape as it encounters it. At the same time, each time a variable name 1^r is encountered, the stack automaton can check if the r-th bit from the top of the stack is 1 or 0. In its finite control, the machine can determine if each clause contains a literal which evaluates to true. If, at any time, the machine discovers that x is not in the proper form, or that some clause contains no literal which evaluates to true, then the machine can make the clause currently under consideration trivially true (e.g. by adding $a \vee \neg a$) and then scan the rest of the input producing no output. The stack automaton constructed in this way computes a function with SAT as its range. By modifying the encoding somewhat, one can construct a real-time machine computing a function whose range is also NP-complete. □

Theorem 4.9 is strong evidence that there are no significant "natural" classes of machines which compute only invertible functions other than the classes of machines given by Theorem 4.3. We considered a large number of the types of automata which have been discussed in the literature. In every case, if the class of machines was not a subclass of one of C_1, C_2, or C_3 of Theorem 4.3, then the class contained one of the classes D_1 through D_8.

Let us now discuss the possibility that stronger results might be obtained if only one-one functions are considered.

The class U was defined and discussed by Valiant, Berman, and others [Be-77, Va-76, GS-84]. A language is in U iff it as accepted by a nondeterministic Turing machine in polynomial time which has at most one accepting computation on any input. U should not be confused with the class of problems, such as Unique-SAT, which consist of those elements of a set in NP which are accepted by exactly one accepting computation. $U \subseteq NP$, whereas Unique-SAT is not believed to be in NP.

A set L is in U iff L is the range of an honest function which is computable in polynomial time and which is one-one on some domain in P [Va-76].

For most of the classes $\mathbf{D}_i$ discussed in Theorem 4.9, it is true that for every set L_1 in U there is a domain L_2 in P and an honest function f computed by a machine in $\mathbf{D}_i$ which is one-one on L_2, such that $L_1 = f(L_2)$. Thus if every honest function which computed by a machine in $\mathbf{D}_i$ which is one-one on a set L in P could be easily inverted on L, it would follow that P = U, which seems unlikely.

Classes of Functions with Inverses in SC

Since Theorem 4.3 showed that functions which are easy enough to compute have inverses in NC and PUNC, it is natural to ask if those inverses are also in SC or DLOG. In this section we present some classes of functions with very easy inverses, but we also show that a class of machines must be *very* simple indeed to compute only functions with inverses in SC.

Theorem 4.10:

(a) There is an honest function computable by a two-way DFA whose range is complete for NLOG under logspace reductions.

(b) There is an honest function computed by a one-way, one-counter machine whose range is complete for NLOG under logspace reductions.

(c) There is an honest function computed by a one-way, one-turn PDA whose range is complete for NLOG under logspace reductions.

Thus, unless NLOG $\subseteq$ SC, there are honest functions in these classes which have no inverse which is in SC.

Proof: (a) Let L be the set of encodings of graphs with a start vertex and a goal vertex , such that there is a path in the graph from the start vertex to the goal vertex which goes from left to right (in the encoding of the graph). L is complete for NLOG under logspace reductions [HM-81]. A two-way DFA can be constructed which has two tracks on its input tape, where the top track contains an encoding of a graph and the bottom track contains markers under certain edges in the graph. The DFA starts scanning its input and printing the description of the graph on its output tape. When it encounters a marked edge (v,w), it backs up to the previous marked edge (x,y) (or it backs up to the start vertex if there is no previous marked edge). Instead of writing (v,w) on its output tape, it writes (y,w) (or (s,w), where s is the start vertex). It then returns to (v,w) and continues scanning the input and copying the input to the output tape. If at any time the DFA discovers that the input tape is not in the proper format, it writes an edge from the start vertex to the goal vertex on its output tape.

(b) and (c) It can be shown, as in [Vi-81] or [AHU-69], that a set L is the range of an honest function computed by a deterministic automaton with a one-way input head iff L is accepted by a nondeterministic one-way automaton of the same type. It was shown in [Su-75a] that there is a language accepted by a one-way nondeterministic counter machine which is complete for NLOG, and it was shown in [Su-75b] that there is a linear CFL which is complete for NLOG. □

Theorem 4.10(a) improves a result in [Gr-78b], where it was noted that the range of any two-way DFA transduction is in NLOG, but it was left open whether or not every such range was in fact in DLOG. Because of 4.10(a), we see that any class of machines which allows two-way or even 3-crossing bounded access to the input includes machines which compute functions which have inverses which are hard for NLOG. In addition, one-way

machines which have counters or reversal-bounded pushdowns for storage also can compute functions with inverses which are hard for NLOG. We do not know if one-way machines with reversal-bounded counters compute only functions with inverses in SC. If that were the case, then it would follow that languages accepted by one-way nondeterministic machines with one single-turn counter are in SC.

In spite of Theorem 4.10, we are able to prove the following positive results:

Theorem 4.11:

(a) If f is computable by a one-way Turing machine with a worktape preset to length loglog n, then f has an inverse in SC.

(b) If f is computable by a one-way DFA, then f has an inverse computable by a two-way DFA.

Proof: (a) The proof of part (a) is very similar to the proof of Theorem 4.3(b). Since there are at most $2^{O(\log\log n)} = \log^{O(1)} n$ worktape configurations, there is no need to use nondeterminism to choose among the possible computations. The details are omitted.

A corollary of Theorem 4.11(a) is that the class of languages accepted by nondeterministic one-way loglog n space-bounded machines is in SC. However, it was already known that one-way alternating loglog n space-bounded machines accept only sets in SC [Su-83] and that NSPACE (loglog n) $\subseteq$ SC [MS-80].

(b) For one-one functions, this was proved in [CJ-77]. (See also [DKR-76, DKR-82].) We will also use the fact that the range of any 1DFA transduction is regular [AU-72]. Here, we use the definition of [CJ-77] that a 1DFA transduction is only defined on the set of those x such that the 1DFA ends up in an accepting state after scanning x.

Now let f be any 1DFA transduction, computed by the 1DFA M. Let M have q states.

Let C = $\{w \mid$ M, on input w, never makes q moves without producing output$\}$. Note that for all $z \in$ range(f), $f^{-1}(z) \cap$ C is a nonempty finite set. C is regular.

There is a 1DFA transduction g such that the range of g is the set $A = \{w \in C \mid$ for some $x \in C$, $x \neq w$, and $f(x) = f(w)\}$. The input to the transducer has w written on one track and x written on aother track. w and x agree on some (possibly empty) prefix; w and x are written one character per cell on that prefix. After that prefix, i.e., after the first cell in which w and x differ, w and x are written so that, in any given cell, the same number of bits of output have been produced by the 1DFA on the portion of w which has been seen as on the portion of x which has been seen. The 1DFA can also check that at no point in the processing of w or x, q moves are made without producing output; thus both w and x are in C.

Let us say that $y \lhd w$ iff either y is a proper prefix of w or $w = w_1 1 w_2$ and $y = w_1 0 y_2$. Clearly, $y \neq w \Rightarrow y \lhd w$ or $w \lhd y$.

There is a 1DFA transduction h so that the range of h is the set $B = \{w \in A \mid$ for some $y \in A$, $f(w) = f(y)$ and $y \lhd w\}$ The construction of the 1DFA computing h is similar to that of the 1DFA computing g.

Let $D = (C - A) \cup (A - B)$. Note that D is regular. We claim that $|f^{-1}(z) \cap D| = 1$ for all $z \in$ range(f).

First note that $f^{-1}(z) \cap C$ is nonempty, and

$$\begin{aligned} f^{-1}(z) \cap C &= f^{-1}(z) \cap ((C - A) \cup (A \cap C)) \\ &= f^{-1}(z) \cap ((C - A) \cup A) \\ &= (f^{-1}(z) \cap (C - A)) \cup (f^{-1}(z) \cap A) \end{aligned}$$

If $f^{-1}(z) \cap (C - A) \neq \varnothing$ then let $w \in f^{-1}(z) \cap (C - A)$. By definition of A, there is no other $x \in C$ such that $f(x) = f(w) = z$, so $|f^{-1}(z) \cap C| = |f^{-1}(z) \cap D| = 1$.

If $f^{-1}(z) \cap (C - A) = \varnothing$, then $f^{-1}(z) \cap A \neq \varnothing$. Assume that $|f^{-1}(z) \cap (A - B)| \geq 2$; let x and w be distinct elements in $f^{-1}(z) \cap (A - B)$. Assume without loss of generality that $x \lhd w$. But then $w \in B$, which is a contradiction. Thus $|f^{-1}(z) \cap (A - B)| \leq 1$.

Assume that $|f^{-1}(z) \cap (A - B)| = 0$. Then for every x_i in $f^{-1}(z) \cap A$ there is an $x_{i+1} < x_i$ in $f^{-1}(z) \cap A$. (If there were no such x_{i+1}, it would follow that $x_i \in B$ and thus $x_i \in f^{-1}(z) \cap (A - B)$.) But the existence of the sequence $x_1, x_2, \ldots$ contradicts the fact that $f^{-1}(z) \cap A \subseteq f^{-1}(z) \cap C$ is finite. Thus $|f^{-1}(z) \cap (A - B)| = 1$.

Now modify M so that it rejects if its input is not in D. The resulting 1DFA transduction is one-one on D, and has range equal to range(f). By [CJ-77], the new 1DFA transduction has an inverse computable by a two-way DFA. □

Classes of Circuits

The techniques used in the preceding sections also allow us to show that certain interesting classes of circuits compute only invertible functions. The circuit complexity measures which have received the most attention are circuit size, depth, and width; see e.g. [Pi-79] for definitions. The usual notion of circuit width allows each input to be available any number of times at no extra cost. Although this definition of circuit width allows one to prove appealing results [Pi-79], it is arguably less natural than the notion of width which allows each input to be available only once. If we think of narrow circuits as being circuits of width $O(\log n)$, where each input is available only once, the next result says that shallow circuits compute inverses of functions computed by narrow circuits.

Theorem 4.12:

Let f be an honest function computed by a family of circuits $\{C_i\}$ of polynomial size and width $O(\log n)$, where each input is available only once in each circuit. Then there is an inverse of f in NC.

Proof: The computation of such circuits is very much like the computation of one-way logspace-bounded machines. The proof of Theorem 4.12 is very similar to the proof of Theorem 4.3(b). □

It is worth noting that Theorem 4.12 is "uniform" in the sense that there is a single polylog-time P-RAM which, given as input y and a suitably-encoded description of a circuit of width $k \log n$, will find an x such that the circuit maps x to y, if one exists. Contrast this with the situation of depth-one circuits of linear size, indegree $\leq$ three, consisting only of "or" gates; it is easy to show that the problem of uniformly inverting functions computed by such circuits is NP-complete, although we are unaware of any such functions having NP-complete range. There are functions computed by depth-two polynomial-size *unbounded*-fan-in circuits having NP-complete range. Thus there seems to be no chance of proving a converse to Theorem 4.12.

CHAPTER V

Ranking Functions

Theorem 4.3 leads to unexpected results about ranking functions and census functions. Given a language L, the ranking function for L is $R_L(w) = |\{x \mid x \in L \text{ and } x \leq w\}|$. The census function for L is $C_L(n) = R_L(1^n)$ = the number of strings in L of length no more than n. Census functions are considered in [HM-80, Ma-82]; they were helpful in proving that no sparse set can be NP-complete if P $\neq$ NP.

We show that any language accepted by a machine in C_1, C_2 or C_3 has an easy ranking function (and hence an easy census function).

Similar results about ranking functions were obtained independently by Goldberg and Sipser [GS-85] in an investigation of classes of languages for which data compression is possible. If a language has an easy ranking function, then it can be "compressed" maximally in the sense that there is a one-one, invertible map taking L onto Σ^*.

Theorem 5.1:

Let L be a language accepted by a machine in C_1, C_2, or C_3. Then $R_L(n)$ is computable in polynomial time.

Proof: Let L be accepted by a machine M in C_2, or C_3. Let M′ be the machine constructed in the proof of Theorem 4.1 which accepts the range of the function f computed by M. Let M_1 be a nondeterministic logspace-bounded Turing machine that, on input y, begins by marking off $\log |y|$ on its worktape, and then simulating M′ on input 1 (so that M_1 will have an accepting computation for every string x of length $\leq |y|$ such that M outputs 1 on input x). In addition,

we require that M′ accept only if the input string it "guesses" comes before y in the standard lexicographic ordering. M_1 will have exactly one accepting computation for each string $w \leq y$ such that $w \in L$.

Since M_1 is logspace-bounded, we can write down the polynomially-many configurations of M_1, and mark each configuration which appears in an accepting computation tree. In polynomial time, we can compute the number of accepting computations which are rooted at any given configuration C: call this *count*(C). If C is an accepting configuration (a leaf), there is one accepting computation rooted at C, so *count*(C) = 1. If C is not a leaf then the number of accepting computations which are rooted at C is Σ *count*(D) where the sum is taken over all configurations D such that $C \vdash D$.

Since there are no configurations C and D of M_1 such that $C \vdash^* D \vdash^* C$, the following procedure computes *count* for each configuration.

> Compute a topological sort on the configurations, so that if D comes before C, it is not the case that $C \vdash^* D$.
> Compute *count*(C) for all configurations C, processing the configurations in reverse order (so that if $C \vdash D$, then D is processed before C).

If C is the initial configuration, then $R_L(y)$ = *count*(C).

If L is accepted by a machine in C_1, we construct the machine M_1 from M′ exactly as above. On input $y\#h(|y|)$, there will be exactly one accepting computation tree of M_1 for each string $w \leq y$ such that $w \in L$.

Since M_1 is logspace-bounded, we can write down the polynomially-many configurations of M_1, and mark each configuration which appears in an accepting computation tree. Let *count* be defined as above. If E is an accepting configuration (a leaf) of M_1, *count*(E) = 1. Each such configuration is attempting to verify that some pair (C,D) is a realizable pair, where the input and output head positions are the same in C as in D. For each non-leaf configuration E of M_1 which is attempting to verify that (C,D) is a realizable pair *count*(E) can be computed in the following way:

count := 0
For each surface configuration A
    If (C,A) and (A,D) are both realizable pairs, let E_1 be the successor of E which verifies that (C,A) is a realizable pair, and let E_2 be the successor of E which verifies that (A,D) is a realizable pair.
    count := *count* + *count*(E_1)**count*(E_2)
For each realizable pair of surface configurations (A,B)
    If C $\vdash$ A via a push of Γ and B $\vdash$ C via a pop of Γ, let E_1 be the successor of E which verifies that (A,B) is a realizable pair.
    count := *count* + *count*(E_1)

Since there are no configurations C and D of M_1 such that C $\vdash^*$ D $\vdash^*$ C, the following procedure computes *count* for each configuration.

Compute a topological sort on the configurations, so that if D comes before C, it is not the case that C $\vdash^*$ D.
Compute *count*(C) for all configurations C, processing the configurations in reverse order (so that if C $\vdash$ D, then D is processed before C).

If C is the initial configuration, then $R_L(y)$ = *count*(C). □

It is pointed out in [GS-85] that there are some sets which are very easy to recognize which have ranking functions which are hard for #P, and thus are as hard as any ranking function for a set in NP. However it is tempting to speculate that there is some connection between the complexity of a set and the complexity of its ranking function.

For instance, every set in DLOG can be reduced by a one-one invertible logspace reduction to a set recognized by a one-way logspace-bounded Turing machine [HM-81]. The same proof technique shows that any set which is recognized by a deterministic AuxPDA which moves its input head $n^{O(1)}$ times can be reduced by a one-one invertible logspace reduction to a set recognized by a one-way deterministic AuxPDA. (The class of sets recognized by deterministic AuxPDA's which moves their input heads $n^{O(1)}$ times is the P-uniform analog of the Sudborough's class log(DCFL) [Su-78]. More is said on this subject in Chapter 6.) By Theorem 5.1, any set in either of these classes is reducible via a one-one invertible logspace reduction to a set with an easy ranking function. We suspect that the same is not true for every set in P.

A number of results have been proved which point to a sort of "duality" which exists between SC and NC. The following result points to a different aspect of that duality.

Theorem 5.2:

Every set in SC can be reduced by a one-one invertible logspace reduction to a set with a ranking function which can be computed in time $n^{\log^{O(1)}n}$.

Every set in PUNC can be reduced by a one-one invertible reduction which can be computed in $\log^{O(1)}n$ space to a set with an easy ranking function.

Proof: Every set in SC can be reduced by a one-one invertible logspace reduction to a set accepted by a real-time, $\log^{O(1)}n$ space-bounded Turing machine with a one-way input tape [Su-83]. Ranking functions for sets accepted by such machines can be computed in time $n^{\log^{O(1)}n}$, via a straightforward extension of the method of Theorem 5.1.

Also, every set in SC can be reduced by a one-one invertible logspace reduction to a set accepted by a one-way loglogspace-bounded alternating Turing machine [Su-83]. Ranking functions for sets accepted by one-way loglogspace-bounded alternating Turing machines can be computed in time $n^{\log^{O(1)}n}$, by a combination of the methods of Theorem 5.1 and Theorem 4.11.

Every set in PUNC can be reduced to a set accepted by a one-way deterministic AuxPDA, using the same techniques as were used in [Su-83], where two-way computations are reduced to one-way computations. The reduction is one-one and can be inverted in polynomial time, and can be computed in $\log^{O(1)}n$ space. The length of the output of the reduction on inputs of length n is $n^{\log^{O(1)}n}$. □

We suspect that no set which is complete for P has a ranking function which is computable in subexponential time, although we have been unable to find very convincing evidence to support this suspicion.

It should be noted that p-isomorphisms do not preserve the property of having an easy ranking function. Recall that a set is a p-cylinder iff it has an easy padding function. All p-cylinders in P are p-isomorphic [Yo-83]. Let A be any set accepted by a one-way logspace-bounded machine such that both A and its complement have padding functions; for instance, let A = $1\{0,1\}^*$. Let B be any set with a hard ranking function; for instance let B be the set given in [GS-85]. Then the set C = $\{x\#y \mid x \in A \text{ and } y \in B\}$ is p-isomorphic to A, but has a hard ranking function; $R_B(y) = R_C(1\#y) - R_C(1\#0^{|y|}) + \Sigma_{j<|y|}(R_C(1\#1j) - R_C(1\#0j))$.

CHAPTER VI

The Complexity of Sparse Sets in P

Theorem 3.7 showed that the complexity of sets in PUNC is closely tied to the complexity of the class of tally languages in P. Tally languages have often been studied in conjunction with sparse sets, and it natural to investigate the complexity of sparse sets in P in relationship to the complexity of sets in PUNC.

One subclass of the sparse sets in P, the P-printable sets, is of particular interest in this regard. P-printable sets were defined in [HY-84]; a set S is P-printable iff the function $n \rightarrow \{w \in S \mid n \geq |w|\}$ is computable in polynomial time. The P-printable sets are "effectively sparse" in the sense that it is easy to find all of the elements of S of a given length. It is believed that not all sparse sets in P are P-printable; in this section we present some sufficient conditions for the existence of sparse sets in P which are not P-printable. We also present a new characterization of the P-printable sets.

Since all tally languages in P are in PUNC, it is natural to ask if all sparse sets in P are in PUNC. There seems to be no reason to believe that is the case. We note that the existence of a sparse set in NP but not in PUNC implies $P \neq NP$.

Theorem 6.1: EXPTIME = NEXPTIME $\Leftrightarrow$ all sparse sets in NP are in PUNC.

Proof: ($\Leftarrow$) If all sparse sets in NP are in PUNC, then all tally languages in NP are in P, which implies that EXPTIME = NEXPTIME by [Bo-74].

($\Rightarrow$) In [HY-84] it was shown that EXPTIME = NEXPTIME implies that all sparse sets in NP are P-printable. But all P-printable sets are clearly in PUNC. □

The proof of Theorem 6.1 gives one context in which P-printable sets arise. They also arise in the study of data compression. Sparse sets are clearly good candidates for data compression, as studied in [GS-85]. However, the only general deterministic technique for data compression which was isolated in [GS-85] involves using ranking functions, as discussed in Chapter 5. It is easily shown that the sparse sets with easy ranking functions are precisely the P-printable sets.

P-printable sets are also closely related to concepts which were discussed in Chapter 3. For instance, every P-printable set gives rise to a P-printable sequence, and vice-versa. P-Uniform circuits are therefore also closely-related to P-printable sets.

The only sparse sets which are easily shown to be in PUNC are the P-printable sets. It is reasonable to ask if, in fact, *all* sparse sets in PUNC are P-printable. Although we are not able to answer that question (and we do not believe even that all sparse sets in DLOG are P-printable), we can come close; we are able to give the following machine-based characterization of the P-printable sets.

Theorem 6.2:

L is P-printable iff L is sparse and L is accepted by a nondeterministic one-way logspace-bounded AuxPDA.

Proof: ($\Rightarrow$) Assume that in time polynomial in n one can enumerate the elements of L of length $\leq n$. Then, just as in the proof of Theorem 3.1, a logspace-bounded AuxPDA can obtain the bits of that enumeration one-by-one. Thus a one-way nondeterministic AuxPDA can guess the length n of the input, guess that the input is the r-th string to appear in the enumeration, and then obtain the bits of the enumeration one-by-one and check that the input agrees with the r-th string in the list.

(We note that the result can be made stronger, in that a *deterministic* one-way AuxPDA with log n space marked out on its worktape can accept L, as can a deterministic

one-way AuxPDA which accepts in logspace all inputs in L, but may use unbounded space for inputs not in L.)

($\Leftarrow$) Let L be a sparse set accepted by a nondeterministic one-way logspace-bounded AuxPDA M. Let M′ be the alternating Turing machine constructed in the proof of Theorem 4.3. Let M_1 be the alternating machine which on input $0^n\#h(n)$ marks off log n space on its worktape and simulates M′ on input $1\#h(n)$. Each accepting computation tree of M_1 corresponds to a string $x \in L$ of length $\leq n$. There are at most $n^{O(1)}$ such strings x.

Unfortunately, since M is nondeterministic, each string x may correspond to many accepting computation trees of M_1. Thus we cannot examine each accepting computation tree in $n^{O(1)}$ time.

Since M_1 is logspace-bounded, we can write down the polynomially-many configurations of M_1, and mark each configuration which appears in an accepting computation tree. Now consider a configuration E which verifies that (C,D) is a realizable pair; let i_C and i_D be the input head positions of M represented in C and D, respectively. Each accepting subtree rooted at E corresponds to a substring which appears in positions i_C through i_D in some string of length n in L. Although the number of accepting subtrees rooted at E may not be polynomially-bounded, the number of corresponding substrings *is* polynomially-bounded, and a dynamic programming algorithm can be used to associate that set S(E) of strings with E.

$S := \varnothing$
For each surface configuration A
    If (C,A) and (A,D) are both realizable pairs, let E_1 be the successor of E which verifies that (C,A) is a realizable pair, and let E_2 be the successor of E which verifies that (A,D) is a realizable pair.
    $S := S \cup S(E_1) \times S(E_2)$
For each realizable pair of surface configurations (A,B)
    If $C \vdash A$ via a push of Γ and $B \vdash C$ via a pop of Γ, let E_1 be the successor of E which verifies that (A,B) is a realizable pair.
    $S := S \cup S(E_1)$

Since there are no configurations C and D of M_1 such that $C \vdash^* D \vdash^* C$, the computation outlined above can proceed very much in the same manner as in the analogous algorithm in Theorem 5.1. □

Theorem 6.2 is interesting because one-way AuxPDA's are not very powerful machines; in [Br-77a], counting arguments were used to show that some relatively "natural" languages in P are not accepted by one-way AuxPDA's of sublinear space complexity. However, any argument showing that no one-way AuxPDA accepts some *sparse* set in P (or even in PSPACE) is also strong enough to settle several outstanding problems in complexity theory, since e.g. P = PSPACE ⇒ all sparse sets in P are P-printable.

Corollary 6.3:

EXPTIME = NEXPTIME ⇔ all sparse sets in NP are accepted by one-way logspace-bounded AuxPDA's.

Proof: Immediate from Theorem 6.2 and the proof of Theorem 6.1. □

In [Su-78], Sudborough investigated the classes log(DCFL) and log(CFL), the classes of languages logspace-reducible to (deterministic) context-free languages. He showed that log(DCFL) and log(CFL) are the classes of languages accepted by deterministic and nondeterministic logspace-bounded AuxPDA's in polynomial time, respectively. It is not known if every sparse CFL is in log(DCFL). (Note that Ruzzo showed in [Ru-81] that every CFL, and in fact every set in NC, is accepted by a deterministic logspace-bounded AuxPDA in time $2^{\log^{O(1)} n}$.)

The class of languages accepted by deterministic (nondeterministic) logspace-bounded AuxPDA's which move their input heads at most $n^{O(1)}$ times is the P-uniform analog of the class log(DCFL) (log(CFL)). By Theorem 6.2, every sparse set accepted by a one-way nondeterministic AuxPDA can be accepted by a deterministic AuxPDA which moves its input head $O(n)$ times.

The question of whether or not sparse sets in P exist which are not P-printable is related to the question of whether or not one-way functions exist.

Theorem 6.4:

If there is a P-printable set S and an honest one-one function f which is computable in polynomial time but is hard to invert on S, then $f^{-1}(S)$ is a sparse set in P which is not P-printable.

Proof: Let f and S be as in the statement of the theorem. If the function $n \rightarrow \{x \in f^{-1}(S) \mid n \geq |x|\}$ were computable in polynomial time, then given any element $y \in S$, $f^{-1}(y)$ could be computed on S by writing out $\{x \in f^{-1}(S) \mid p(|y|) \geq |x|\}$, applying f to each element of $\{x \in f^{-1}(S) \mid p(|y|) \geq |x|\}$, and seeing if any x maps to y. Thus f would be easy to invert on S, which is a contradiction. □

If the set S in the statement of Theorem 6.4 is in DLOG and f is computable in logspace, then $f^{-1}(S) \in$ DLOG. We conjecture that sparse sets exist in DLOG which are not P-printable.

The existence of sparse sets in P which are not P-printable does not seem to imply the existence of one-way functions; that is, the converse to Theorem 6.4 does not seem to be true.

In [Va-76], Valiant addressed the question of whether or not "evaluating" was harder than "checking." This has also been discussed in, e.g., [Li-85]. One aspect of the evaluating vs. checking question has relevance to the topics covered in this chapter. Let a sequence s_1, s_2,... be *easy to evaluate* if the function $n \rightarrow s_1, s_2, \ldots s_n$ can be computed in polynomial time, and let it be *easy to check* if the set $\{s_1\#s_2\# \ldots\#s_n \mid n \geq 1\}$ is in P. (Note that the set $\{s_1\#s_2\# \ldots\#s_n \mid n \geq 1\}$ is a set of strings which encode the first n elements of the sequence. This is different from requiring that the set $\{s_1, s_2, \ldots s_n \mid n \geq 1\}$ is in P.) To avoid the possibility that a sequence could be easy to check but hard to evaluate for trivial reasons, we also require that $|s_n| = n^{O(1)}$ for all n.

Theorem 6.5:

There is a sequence which is easy to check but hard to evaluate ⇔

there exists a polynomial-time computable function which is honest, one-one, and hard to invert on a P-printable set.

Proof: (⇒) Let $s_1, s_2, \ldots s_n$ be easy to check but hard to evaluate. Then the function $f(x) = 0^n$ if $x = s_1\#s_2\# \ldots\#s_n$, and $f(x) = 1x$ is one-one, honest, and computable in polynomial time. Inverting f on 0^* is equivalent to evaluating the sequence $s_1, s_2, \ldots s_n$.

(⇐) Let f be a polynomial-time computable function which is honest, one-one, and hard to invert on a P-printable set $\{s_1, s_2, \ldots s_n \mid n \geq 1\}$. The sequence $f^{-1}(s_1), f^{-1}(s_2), \ldots$ is easy to check; given input $t_1\#t_2\# \ldots\#t_n$, simply compute $f(t_1)\#f(t_2)\# \ldots\#f(t_n)$, and compare the result against $s_1\#s_2\# \ldots\#s_n$. This is easy since $\{s_1, s_2, \ldots s_n \mid n \geq 1\}$ is P-printable. If $f^{-1}(s_1)$, $f^{-1}(s_2), \ldots$ were easy to evaluate, then, just as in the proof of Theorem 6.4, f^{-1} would be easy to evaluate on $\{s_1, s_2, \ldots s_n \mid n \geq 1\}$. □

CHAPTER VII

Precomputation

In Chapter 3, we presented a characterization of PUNC in terms of general-purpose parallel computers, such as SIMDAGs, which are augmented with a P-printable sequence. The P-printable sequence is used to provide the SIMDAG with access to information which has been precomputed. In this chapter we investigate using P-printable sequences to model precomputation in other settings.

When augmenting a space-bounded computation with precomputed information, it makes sense to bound the length of that information. That is, if one wants to store a table of precomputed information which will help in evaluating a function on all inputs up to some length n, it may well be that considerably less than n memory locations are going to be available to store the table; thus one would want to investigate using a smaller table. Here, we will consider what the proper way is to model supplying only a small number of precomputed bits.

We first present one answer to this question, then show that it is not satisfactory, and then present a new answer and show that characteristic sequences again give an elegant characterization of precomputation.

Computations augmented by short strings of "nonuniform" data were studied in [KL-82], as a generalization of nonuniform circuit complexity. The principal difference between the definitions of [KL-82] and the definitions considered here is that the "advice" bits of [KL-

82] did not need to be effectively or efficiently constructible. Here, we are trying to model feasible precomputation.

Translating directly from the definition of PUNC, and borrowing from the notation of [KL-82], one can formulate the following definition.

Definition A: SC/feasible$_A$-O(log$^k n$) is the class of all languages L such that there is a function $n \rightarrow h(n)$ computable in time polynomial in n, where $|h(n)| \leq l\log^k n + l$ for some l, and there is some language $L' \in$ SC such that for all strings w, $w\#h(|w|) \in L'$ iff $w \in L$.

The problem with definition A is that the precomputed help $h(n)$ is only required to work for inputs of length n. A table storing help for *all* input lengths $\leq n$ would thus need to be of length at least n, whereas we would like for such a table to be small: of length $\log^{O(1)} n$. That motivates the following definition.

Definition B: SC/feasible-O(log$^k n$) is the class of all languages L such that there is a function $n \rightarrow h(n)$ computable in time polynomial in n, where $|h(n)| \leq l\log^k n + l$ for some l, and there is some language $L' \in$ SC such that for all strings w of length $\leq n$, $w\#h(n) \in L'$ iff $w \in L$.

Conjecture: $\{0^n \mid n \in U\}$ is in SC/feasible$_A$-O(log$^k n$) but not in SC/feasible-O(log$^k n$).

A *Turing machine augmented with O(f(n)) bits of sequence s* is a Turing machine with the first $f(n)$ bits of the sequence s written on one of its worktapes.

Let $V_k = \{M(00)^s 1 w \mid M \text{ accepts } w \text{ in time } 2^{2^{(s+|w|)^k}}\}$, where the encoding of machines M is such that no encoding contains "00" as a substring. (Any set which is complete for DTIME($2^{2^{n^k}}$) under polynomial time reductions f such that $|f(x)| = |x| + O(1)$ can be substituted for V_k.) The first O(log$^k n$) bits of s_{V_k} may be obtained in time $n^{O(1)}$. (To obtain the r-th bit, see if $r \in V$ in time $2^{O(2^{|r|/k})}$; for $r \leq l\log^k n + l$, $|r|/k \leq \text{loglog } n + O(1)$, and thus this takes time $n^{O(1)}$.)

Theorem 7.1:

L is in SC/feasible-O($\log^k n$) iff L is accepted in polynomial time and $\log^{O(1)} n$ space on a Turing machine augmented with O($\log^k n$) bits of s_{V_k}

Proof: ($\Leftarrow$) Let L be accepted in polynomial time and $\log^{O(1)} n$ space on a Turing machine augmented with $l\log^k n + l$ bits of s_{V_k}. Let $h(n)$ be the first $l\log^{O(1)} n + l$ bits of s_{V_k}. It was observed above that $n \rightarrow h(n)$ is computable in time polynomial in n. It is now easy to construct the desired language L′ in SC.

($\Rightarrow$) Let $|h(n)| \leq l\log^k n + l$. The language $L_1 = \{x \mid \log(l(\log 2q-1)^k + l + 1) \leq |x| < \log(l(\log 2q)^k + l + 1), x = 0^i 1 r$ for some $i \geq 0$, and the r-th bit of $h(2q)$ is 1$\}$ can be recognized in time $(2q)^{O(1)} = (2^{2^{|x|/k}})^{O(1)}$; let M recognize L_1 in time $2^{2^{(s+n)/k}}$. To recognize L, on input w simulate the SC machine accepting $w\#h(2q)$, where $2q-1 \leq |w| < 2q$. To obtain the r-th bit of $h(2q)$, set $x = 0^i 1 r$ where $|x| = \log(l\log^k |w| + l + 1)$, and consult the $M(00)^s 1x$-th bit of s_{V_k}. Because $|x|$ is small, no more than O($\log^k n$) bits of s_{V_k} need to be consulted. □

The significance of Theorems 3.6 and 7.1 is that there is a single "universal" sequence of advice bits which can be used to model all feasible precomputation. This should be constrasted with the case of nonuniform advice of the sort studied in [KL-82]. For instance, let L be a language in P/*poly*, the class of languages (defined in [KL-82]) which can be solved in polynomial time relative to "advice bits" for inputs of length n, where the length of the advice is bounded by a polynomial in n. Equivalently, let L be accepted by a nonuniform family $\{C_n\}$ of circuits of size polynomial in n. Then there is a set which can not be accepted *in any amount of time* relative to $\{C_n\}$: namely any set which is not r.e. in $\{C_n \mid n \in \mathbb{N}\}$. Thus there is no "universal" sequence of nonuniform advice bits.

We note in concluding this section that Turing machines augmented with sequences can also be used to characterize PUNC in terms of sequential computation; a language L is in PUNC iff L is accepted in time $n^{O(1)}$ and reversal $\log^{O(1)} n$ by a Turing machine augmented with s_U [Pi-79].

CHAPTER VIII

The Structure of Complete Sets

The results of Chapter 4 about invertibility, combined with the results of Chapter 5 about ranking functions, enable us to prove some results about the structure of complete sets.

1-L reductions, which are functions computable by logspace-bounded Turing machines which have a one-way input head, were introduced in [HIM-78] as a tool for determining relationships between complexity classes which are too fine to be detected using Karp reductions or logspace reductions. For example, there are natural sets which are complete for DLOG under 1-L reductions, whereas it makes little sense to speak of sets complete for DLOG under logspace reductions. It is observed in [HIM-78] that most NP-complete problems which have appeared in the literature (in fact, *all* of the "natural" NP-complete problems which the authors of [HIM-78] considered) are complete for NP under 1-L reductions. (It is easy, however, to *construct* a set p-isomorphic to SAT which is not complete under 1-L reductions [HIM-78].)

In [HM-81] it is shown that no set complete under 1-L reductions can be sparse. We greatly improve on that result; we show that all such sets are "almost" p-isomorphic.

(As defined in [HIM-78, HM-81], 1-L reductions are computed by machines which begin the computation with logspace marked off on their worktapes. We will use that definition in this section.)

In Chapter 1 we reviewed work relating to the Berman-Hartmanis conjecture that all NP-complete sets are p-isomorphic. In particular, we saw that a set is p-isomorphic to SAT iff

it is complete for NP under one-one, length-increasing, invertible Karp reductions. In [Yo-83, JY-85], sets are presented which are complete under one-one length-increasing (but presumably non-invertible) reductions, and which seem not to be p-isomorphic to SAT. The result below shows that sets complete under 1-L reductions are complete under length-increasing, invertible and "almost" one-one Karp reductions, and thus they are "almost" p-isomorphic. Showing that they are in fact p-isomorphic could be a first step toward showing that the Berman-Hartmanis conjecture fails only if one-one one-way functions exist.

A function g will be said to be a *strong inverse* of f if $g(y) = \{x \mid f(x) = y\}$. If g is easy to compute, we say that f is *strongly invertible*. Note that if f is strongly invertible, then f is almost one-one, in the sense that for some polynomial p, $|\{x \mid f(x) = y\}| \leq p(|y|)$.

Theorem 8.1:

Let A be complete for NP (or DLOG, NLOG, P, etc) under 1-L reductions. Then A is complete under length-increasing, strongly-invertible Karp reductions.

In order to prove this theorem we will need to prove a series of lemmas.

Lemma 8.2:

There is a set SAT′ p-isomorphic to SAT such that $w \in \text{SAT}' \Rightarrow |w|$ is a power of 2 and $|w| > 1$.

Proof: Let $\text{SAT}' = \{x10^{r-1} \mid r = 2^{\lfloor \log |x| \rfloor + 1} - |x|\}$. Clearly $\text{SAT} \leq_{\text{LIFP}} \text{SAT}'$, and thus SAT and SAT′ are p-isomorphic. □

In what follows, let A be a given set which is complete for NP under 1-L reductions, let $B = \{w^{2|w|} \mid w \in \text{SAT}'\}$, and let f be a 1-L reduction computed by a machine M, where f reduces B to A.

Lemma 8.3: There is a p-printable set S containing all $w \in \text{SAT}'$ such that $|f(w^{2|w|})| \leq |w|$.

Proof:

On input n, the following routine prints a list containing all words $w \in SAT'$ such that $|f(w^{2|w|})| \leq |w| \leq n$.

```
begin
    for m:= 1 to ⌊log n⌋                    //Print all such w with |w| = 2^m.
        (1) Create a labelled digraph G=(V,E) with V being the set of all configurations of M
            of size 1 + 2m (= log|w^{2|w|}|, if |w| = 2^m), and E containing an edge labelled a ∈ Σ ∪
            {ε} from C_i to C_j iff M has a move C_i ⊢ C_j which consumes input a and produces no
            output. (The label l_p of a path p in G is the concatenation of the labels of its edges.)
        (2) for each configuration C_i ∈ V
                Make a copy G_i of G
                By doing a breadth-first search of G_i starting at C_i, find and mark those edges
                which can be traversed by a path p from C_i, where |l_p| ≤ 2^m.
                Delete all unmarked edges and all vertices which are not connected to C_i by a
                marked path.
                for each C_j in G_i
                    if there are two paths from C_i to C_j, mark C_j
                Delete all marked vertices. (G_i is now a tree, since for every C_j in G_i, there is
                                                exactly one path from C_i to C_j.)
                for each C_j in G_i
                    if the path p from C_i to C_j has |l_p| = 2^m, put l_p in TEMP
    for each w ∈ TEMP
        if |f(w^{2|w|})| ≤ |w|, output w.
end
```

To see that the routine is correct, let w be any word in SAT' such that $|f(w^{2|w|})| \leq |w|$. We need to show that the routine outputs w on input $n \geq |w|$.

On input $w^{2|w|}$, M outputs a string of no more than $|w|$ bits. Thus there must exist some $r < 2|w|$ such that after reading w^r, M produces no output while reading the $r+1$-st w. Let C_i be the configuration M enters after consuming w^r, and let C_j be the configuration M enters after consuming w^{r+1}.

Let us assume that the routine does not output w. Then there must be some path p in G_i from C_i to C_j, with $l(p) = x \neq w$. Note that without loss of generality, $|x| \leq |w|$, since all edges corresponding to words of length $> |w|$ are deleted before we check for duplicate paths.

Consider M's computation on input $w^r x w^{2|w|-r-1}$. In order to determine M's initial configuration, we must calculate $\lceil \log |w^r x w^{2|w|-r-1}| \rceil$. Since $w \in \text{SAT}'$, $|w| = 2^m$ for some $m > 0$. Thus

$$\begin{aligned} 2m + 1 = \lceil \log 2^m(2^{m+1} - 1) \rceil &= \lceil \log 2^{2m+1} - 2^m \rceil \\ &= \lceil \log 2|w|^2 - |w| \rceil \\ &\leq \lceil \log 2|w|^2 - |w| + |x| \rceil \\ &\leq \lceil \log 2|w|^2 \rceil = 2m + 1. \end{aligned}$$

That is, M's initial configuration is the same on input $w^r x w^{2|w|-r-1}$ as on input $w^{2|w|}$. Thus on input $w^r x w^{2|w|-r-1}$, M enters configuration C_i after reading w^r, enters C_j after reading x without producing any output while reading x, and then finishes the computation, reading $w^{2|w|-r-1}$. Thus M produces the same output on input $w^r x w^{2|w|-r-1}$ as on input $w^{2|w|}$; thus $f(w^r x w^{2|w|-r-1}) = f(w^{2|w|})$ and $w^r x w^{2|w|-r-1} \in B$, since $w^{2|w|} \in B$. Thus $w^r x w^{2|w|-r-1} = y^{2|y|}$ for some $y \in \text{SAT}'$. Since $|x| \leq |w|$, , we must have $|y| \leq |w|$; however, $|y| = |w|$ implies $y = w = x$, which contradicts $x \neq w$. Thus $|y| < |w|$; however $|y| < |w|$ implies $|y| \leq 2^{m-1}$, which implies $|y^{2|y|}| \leq 2^{2m-1} < 2^{2m+1} - 2^m \leq |w^r x w^{2|w|-r-1}|$, which contradicts $w^r x w^{2|w|-r-1} = y^{2|y|}$. □

Lemma 8.4: There is a polynomial q_1 such that $y \in A \Rightarrow |f^{-1}(y) \cap \Sigma^n| \leq q(n)$ for all n.

Proof: Note that since $y \in A$, we have that $f^{-1}(y) \subseteq B$. We thus have $|f^{-1}(y) \cap \Sigma^n| = 0 \leq q(n)$ unless $n = 2^{2m+1}$ for some m. If $n = 2^{2m+1}$ we may write $f^{-1}(y) \cap \Sigma^n = \{x_1{}^{2m}, x_2{}^{2m}, \ldots, x_r{}^{2m}\}$ for some $r \geq 0$.

M, in its computation on $x_1{}^{2m}$, reaches a point when it has consumed the first x_1 of the input, has output some prefix y' of y, and is in some configuration C_1. We will assume without loss of generality that M keeps track on its worktape of the number of input symbols it has scanned, and thus $|x_1|$ is recorded in C_1.

Since $f(x_i{}^{2m}) = y$ for all i, $1 \leq i \leq r$, M reaches a point in its computation on each $x_i{}^{2m}$ when it has scanned some prefix x_i' of $x_i{}^{2m}$, has output y', and is in some configuration C_i.

If $C_i = C_j$, then

(1) $|x_i'| = |x_j'|$, since M keeps track of the number of input symbols it has consumed.

(2) M outputs y on input $x_i'x_j''$, where $x_j^{2m} = x_j'x_j''$. Note that $|x_i'x_j''| = n$.

(3) $x_i'x_j'' \in B$, and thus $x_i'x_j'' = x_k^{2m}$ for some k, $1 \leq k \leq r$.

Since $|x_k| = m \leq \frac{1}{2}|x_k^{2m}| = \frac{1}{2}|x_i'x_j''|$, x_k is either a prefix of x_i', a suffix of x_j'', or both. Thus $i = k = j$. That is, $C_i = C_j \Rightarrow i = j$.

Thus r is less than or equal to the number of configurations of M on inputs of length n. Since M is a logspace-bounded machine, this number can be bounded by some polynomial q_1.

□

Lemma 8.5: Let preimage(y) = $\{x \mid f(x) = y$ and $|x| \leq |y|\}$. Then |preimage(y)| can be computed in time polynomial in $|y|$, and preimage(y) can be computed in time polynomial in ($|y|$ + |preimage(y)|).

Proof: This is a straightforward generalization of Theorem 5.1, and can be proved using similar techniques.

□

By Lemma 8.4, there is a polynomial q such that if $x \in$ SAT', then $|\text{preimage}(f(x^{2|x|}))| \leq q(|x|)$.

We are now ready to define a procedure which computes a function g which we claim is a strongly-invertible, length-increasing, polynomial-time reduction of SAT' to A.

Since SAT' is p-isomorphic to SAT, there is a one-one, length-increasing, invertible padding function p such that $p(x,y) \in$ SAT' iff $x \in$ SAT'. Let T be some trivial element of SAT'. Let $a(x)$ be the string which differs from $x^{2|x|}$ only in the rightmost bit. Let rejectable(x) and trash(x) be defined by the following procedures.

```
rejectable(x)
begin
    if  [|x| is not a power of two greater than 1] or
        [x ∉ S and |f(x^{2|x|})| ≤ |x|] or
        [|preimage(f(x^{2|x|}))| > q(|x|)] or
        [|preimage(f(x^{2|x|}))| ≤ q(|x|) and there is some element of preimage(f(x^{2|x|})) which is not
        of the form y^{2|y|}]
            then
                return true
            else
                return false
end

trash(x)
begin
    let y be the least such that p(T,x#y) ∉ S
    return f(a(p(T,x#y)))
end
```

The function g is computed by the following routine.

```
g(x)
begin
    if rejectable (x)
        then
            g(x) = trash(x)
        else
            if x ∉ S
                then
                    g(x) = f(x^{2|x|})
                else
                    let y be the least such that p(x,y) ∉ S
                    if rejectable (p(x,y))
                        then
                            g(x) = trash(x)
                        else
                            g(x) = f(p(x,y)^{2|p(x,y)|})
end
```

Lemma 8.6: The routine presented above computes g in polynomial time.

Proof: This is immediate from Lemma 8.5, except for verifying that the operation "let y be the least such that $p(x,y) \notin S$" can be computed in polynomial time. Since S is sparse, at most $|x|^{O(1)}$ elements of $p(x,\Sigma^*)$ need to be examined in order to find some y such that $p(x,y) \notin S$. □

Lemma 8.7: g is a length-increasing reduction of SAT' to A.

Proof: If x is in SAT', then rejectable(x) is false, and $x^{2|x|} \in B$, and $\{f(x^{2|x|}), f(p(x,y)^{2|p(x,y)|}) \mid y \in \Sigma^*\} \subseteq A$. If, in addition, $x \notin S$ (or $p(x,y) \notin S$) then $|g(x)| = |f(x^{2|x|})| > |x|$ (or $|f(p(x,y)^{2|p(x,y)|})| > |p(x,y)| > |x|$).

If x is not in SAT', then if rejectable(x) is true, $g(x) = f(a(p(T,x\#y)))$, which is not in A since $a(p(T,x\#y))$ is not of the form $z^{2|z|}$, and hence is not in B. Since $p(T,x\#y)^{2|p(T,x\#y)|} \in B$, and $p(T,x\#y) \notin S$, M outputs at least one bit while processing each copy of $p(T,x\#y)$ on input $p(T,x\#y)^{2|p(T,x\#y)|}$. Since $a(p(T,x\#y))$ differs from $p(T,x\#y)^{2|p(T,x\#y)|}$ only in the rightmost bit, it follows that $|f(a(p(T,x\#y)))| > |p(T,x\#y)| \geq |x|$.

If x is not in SAT' and rejectable(x) is false, then since x is not in SAT', $x^{2|x|} \notin B$, and $f(p(x,y)^{2|p(x,y)|}) \notin B$ for all y. That g is length increasing and $g(x) \notin A$ follows by an argument similar to that in the preceding paragraph. □

Lemma 8.8: g is strongly-invertible.

Proof: The following procedure computes the strong inverse of g.

```
inverse(y)
begin
    if |preimage(y)| ≤ q(|y|)
        then
            compute preimage(y)
            for each w in preimage(y)
                if w is of the form x^{2|x|} and g(x) = y
                    then
                        put x in LIST
                if w is of the form p(x,u)^{2|p(x,u)|} and g(x) = y
                    then
                        put x in LIST
                if w is of the form a(p(T,<x,u>)) and g(x) = y
                    then
                        put x in LIST
        else
            for each prefix y' of y
                for each configuration C of M of length ≤ ⌈log |y|⌉
                    for each s ∈ {0,1}
                        let y_{C,s} be the string which M outputs when in configuration C with s
                        remaining on the input tape
                        if |preimage(y'y_{C,s})| ≤ q(|y'y_{C,s}|)
                            then
                                compute preimage(y'y_{C,s})
```

```
                    for each w in preimage(y'y_{C,s})
                        if w is of the form p(T,x#u) and g(x) = y
                            then
                                put x in LIST
    output LIST
end
```

To see that the routine is correct, note that if $g(x) = y$, then either $[g(x) = f(x^{2|x|})$ and $|\text{preimage}(f(x^{2|x|})| \leq q(|x|) \leq q(|y|)]$ or $[g(x) = f(p(x,u)^{2|p(x,u)|})$ and $|\text{preimage}(g(x))| \leq q(|x|) \leq q(|y|)]$ or $[g(x) = f(a(p(T,x\#y)))]$. Clearly, the only difficulty arises in the case $g(x) = f(a(p(T,x\#y)))$.

If $g(x) = f(a(p(T,x\#y)))$, then $g(x) = y'z$, where M outputs y' before reading the final input character. The routine tries all prefices y' of y and generates all strings z which M could possibly affix to y', and then checks to see if $p(T,x\#u)^{2|p(T,x\#u)|} \in \text{preimage}(y'z)$ for any u.

It is clear that the running time is polynomial. □

Proof(of Theorem 8.1)**:** Immediate from Lemmas 8.2 through 8.8. □

Given a set L which is complete under one-one 1-L reductions (which are not known to be honest), Theorem 8.1 shows that L is complete under length-increasing, invertible functions which are not necessarily one-one, and are thus not necessarily LIFP. The next result shows that such languages L are in fact complete under LIFP reductions.

Theorem 8.9:

Let A be complete for NP (or DLOG, NLOG, P, etc) under one-one 1-L reductions. Then A is complete under LIFP reductions.

Proof: The function g constructed in the proof of Theorem 8.1 fails to be one-one, since for certain strings x and y we can have $g(x) = g(p(x,y)) = f(p(x,y)^{2|p(x,y)|})$ The function g' computed by the following routine avoids such behavior.

```
g'(x)
begin
    if rejectable (p(x,0))
        then
            g'(x) = trash(x)
        else
```

if $x \notin S$
    then
        $g'(x) = f(p(x,0)2^{|p(x,0)|})$
    else
        let y be the least such that $p(p(x,y),1) \notin S$
        if rejectable $(p(p(x,y),1))$
            then
                $g'(x) = \text{trash}(x)$
            else
                $g'(x) = f(p(p(x,y),1)2^{|p(p(x,y),1)|})$
end

For all strings x, either $g'(x) = f(a(p(T,x\#y)))$, $g'(x) = f(p(x,0)2^{|p(x,0)|})$, or $g'(x) = f(p(p(x,y),1)2^{|p(p(x,y),1)|})$ for some y. Since f is one-one, and since for all $z \neq x$ and all y_1, y_2, y_3, and y_4, $\{a(p(T,x\#y_1)),\ p(x,0)2^{|p(x,0)|},\ p(p(x,y_2),1)2^{|p(p(x,y_2),1)|}\} \cap \{a(p(T,z\#y_3)),\ p(z,0)2^{|p(z,0)|},\ p(p(z,y_4),1)2^{|p(p(z,y_4),1)|}\} = \varnothing$, g' is one-one. The proof that g' is length-increasing and invertible is the same as in the proof of Theorem 8.1 □

For larger complexity classes, Theorem 8.9, together with the techniques of [Be-77] (see also [Do-82, Wa-85]), yield a stronger result.

Theorem 8.10:

All sets complete for PSPACE (DTIME(2^{lin}), DTIME(2^{poly}), DSPACE(2^{lin}), etc.) under 1-L reductions are p-isomorphic.

Proof: We prove only that all sets complete for PSPACE under 1-L reductions are p-isomorphic. The results for other deterministic classes can be proved similarly. This proof is based on the techniques of [Wa-85].

Let M_1, M_2, ... be an indexing of all 1-L machines, and let A be any set complete for PSPACE under 1-L reductions. Let QBF be the set of all satisfiable quantified Boolean formulae; QBF is complete for PSPACE under one-one 1-L reductions [HIM-78].

Consider the set S accepted by a Turing machine M which performs the following computation:

On input z of length n, mark off n^2 space.
If z is not of the form $i\#x\#y10^i$, halt and reject.
Run M_i on input z. (Do not store the output of M_i, but do record how much output M_i produces on input x.) If more than n^2 space is required, halt and reject.

For all $u\#v$ such that $u\#v \leq x\#y$
  Run M_i on input $i\#u\#v10^l$. If more than n^2 space is required, halt and reject.
  Compare the output of M_i on input $i\#u\#v10^l$ with the output of M_i on input $i\#x\#y10^l$. (This comparison can be done bit-by-bit, so that the entire output doesn't need to be stored all at one time.)
  If the output of M_i on input $i\#u\#v10^l$ = the output of M_i on input $i\#x\#y10^l$,
    Then
      Halt and accept iff $u \notin$ QBF
Endfor
(If the computation reaches this point, there is no $u\#v \leq x\#y$ such that the output of M_i on input $i\#u\#v10^l$ = output of M_i on input $i\#x\#y10^l$.)
Halt and accept iff $x \in$ QBF.

Clearly, S $\in$ PSPACE. Thus there is some 1-L reduction computed by some machine M_i reducing S to A. There is some constant r such that for all strings x and y, M can carry out the simulation of M_i on input $i\#x\#y10^r$ in n^2 space. If w = the output of M_i on input $i\#x\#y10^r$ = the output of M_i on input $i\#u\#v10^r$ for some $u\#v \neq x\#y$, then let $u\#v \leq x\#y$ be the lexicographically smallest two strings which map to w in that way. Then $w \in$ A iff $i\#u\#v10^r \in$ S iff $u \in$ QBF, and $w \in$ A iff $i\#x\#y10^r \in$ S iff $u \notin$ QBF. This is a contradiction. Thus the function f which takes $x\#y$ to the output of M_i on input $i\#x\#y10^r$ is a one-one function. Also, f is computable by a 1-L machine. Furthermore, $i\#x\#y10^r \in$ S iff $x \in$ QBF, and thus $f(x\#y) \in$ A iff $i\#x\#y10^r \in$ S iff $x \in$ QBF. That is, f is a 1-L reduction from QBF$\#\Sigma^*$ to A. The function f need not be honest, however.

Since QBF is complete for PSPACE under one-one 1-L reductions, it follows that QBF$\#\Sigma^*$ is complete for PSPACE under one-one 1-L reductions. Since the class of 1-L reductions is closed under composition [HIM-78], it follows that A is complete for PSPACE under one-one 1-L reductions. By Theorem 8.9, A is complete for PSPACE under LIFP reductions, and hence A is a p-cylinder. □

Let us say that the Berman-Hartmanis conjecture is *very false* if there are sets complete for NP under 1-L reductions which are not p-isomorphic. It is known that if the Berman-Hartmanis conjecture is true, then P $\neq$ NP (since if P = NP then finite sets are NP-complete). It follows from Theorem 8.10 that if the Berman-Hartmanis conjecture is very

false, then NP $\neq$ PSPACE. We refrain from conjecturing that the Berman-Hartmanis conjecture is very false.

In [Yo-83], Young posed the question of whether the Berman-Hartmanis conjecture is true iff one-one one-way functions do not exist. A possible first step toward answering this question would be to show that all sets complete for NP under length-increasing, strongly-invertible reductions are NP-complete. We have been unable to show that this is true, even with the additional assumption that the reductions under consideration never map more than two different strings to the same output.

CHAPTER IX

Conclusions and Open Problems

Invertible Functions

There is a certain minimal amount of computational effort which must be put into the computation of a function, if that function is to be hard to invert. Using the concepts of machine- and circuit-based complexity theory, we have attempted to define as precisely as possible the boundary separating classes of computing devices computing only functions which are easy to invert from classes which compute one-way functions. Among the theorems obtained through this investigation are results which show that functions computed by narrow circuits have inverses computed by shallow circuits, as well as results which show that machines with powerful storage structures but limited access to the input compute only easy-to-invert functions, as do machines with unlimited access to the input but only weak storage facilities.

P-Uniform NC

The functions which we show to be easily invertible have inverses which can be computed very quickly in parallel; thus it is unlikely that such functions are hard for P. In order to prove results about the parallel complexity of inverting functions, it was necessary to define a new complexity class, PUNC. Several arguments were presented which support the claim that PUNC models the notion of "feasible parallelism" more successfully than NC, which has been the focus of most complexity-theoretic work on parallelism.

A number of different equivalent characterizations of PUNC were presented in Chapter 3. PUNC can be defined in terms of P-uniform circuits or in terms of alternating Turing machines or AuxPDA's which have limited access to the input. PUNC also has a natural characterization in terms of general-purpose parallel computers.

Other work relating to PUNC remains unfinished. For instance, we believe that a number of theorems in complexity theory can be stated and proved more easily in the context of PUNC than in terms of logspace-uniform families of circuits; doing so has been left to future work. Also, an interesting question is whether or not Pippenger's characterization of NC in terms of time- and reversal-bounded Turing machines can be modified to give a characterization of NC in terms of time- and R-bounded Turing machines, for some "natural" resource R. It is interesting to note in this regard that one-way AuxPDA's have been studied in the past in relation to a "return" complexity measure which is similar in some ways to reversal [BR-77b, Ch-77, We-79, We-80]

Sparse Sets

The study of PUNC has given rise to a number of interesting questions and results about sparse sets in P. A machine-based characterization of the P-printable sets was presented in Chapter 6, and the implications of that characterization were explored. It was shown that the question of the existence of sparse sets in P which are not P-printable is related to the question of the existence of one-way functions and sequences which are easy to check and hard to evaluate. A number of questions about sparse sets in P remain open.

Chapter 6 contains results which indicate that not all sparse sets in P are P-printable; however, non-P-printable sets may still be in PUNC. Are all sparse sets in P in PUNC? Are there any sparse sets for which there is strong evidence that they are not in PUNC? Is there a

sparse set in P which is in PUNC iff every sparse set in P is in PUNC? These questions deserve further attention.

Are all sparse sets in SC in PUNC? Recall in this regard that the exponential-time analog of SC is contained in the exponential-time analog of PUNC.

Ranking Functions

The techniques which enabled us to compute the inverses of easy functions also enabled us to compute ranking functions. A number of questions about ranking functions remain outstanding. Can the class of languages with easy ranking functions be characterized in some way? Can a set which is hard for P have an easy ranking function? What is the complexity of computing ranking functions; e.g., is there a regular set with a ranking function which is hard for P? Finally, there is the intriguing question of whether or not there is any deeper connection than was presented here between the ability to compute ranking functions for languages and the ability to compute inverses for functions.

The Structure of Complete Sets

In Chapter 8, we showed that languages which are complete for complexity classes under very weak reductions are p-isomorphic, or are nearly so. It is interesting to note that the techniques which were used in Chapter 9 are very specific to 1-L reductions. For instance, consider sets which are complete for DTIME($2^{O(n)}$) under two-way DFA reductions. (A number of such sets are presented in [St-74].) It follows from the results of, e.g., [Wa-85], that all such sets are P-isomorphic. However, nothing is known about sets complete for NTIME($2^{O(n)}$) under two-way DFA reductions; it is not even known if such reductions can be replaced by one-one or length-increasing reductions.

REFERENCES

[Ad-79] L. Adleman, *A subexponential algorithm for the discrete logarithm problem with applications to cryptography*, Proc. 20th IEEE Symposium on Foundations of Computer Science, pp. 55-60.

[AHU-68] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Time and tape complexity of pushdown automaton languages*, Information and Control 13, 186-206.

[AHU-69] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *A general theory of translation*, Mathematical Systems Theory 3, 193-221.

[AHU-74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Massachusetts.

[AU-72] A. Aho and J. Ullman, The Theory of Parsing, Translation, and Compiling, Prentice-Hall, Englewood Cliffs, NJ.

[ADP-80] G. Ausiello, A. D'Atri, and M. Protasi, *Structure preserving reductions among convex optimization problems*, J. Computer and System Sciences 21, 136-153.

[BB-74] B. Baker and R. Book, *Reversal-bounded multipushdown machines*, J. Computer and System Sciences 8, 315-332.

[BCH-84] P. W. Beame, S. A. Cook, and H. J. Hoover, *Log depth circuits for division and related problems*, Proc. 25th IEEE Symposium on Foundations of Computer Science, pp. 1-11.

[Be-77] L. Berman, *Polynomial reducibilities and complete sets*, Doctoral Dissertation, Cornell University.

[BH-77] L. Berman, J. Hartmanis, *On isomorphisms and density of NP and other complete sets*, SIAM J. Comput. 6, 305-323.

[BM-84] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudo-random bits*, SIAM J. Comput. 13, 850-864.

[Bo-74] R. V. Book, *Tally languages and complexity classes*, Information and Control 26, 186-193.

[BGW-79] R. V. Book, S. A. Greibach, and C. Wrathall, *Reset machines*, J. Computer and System Sciences 19, 257-276.

[BL-85] R. Boppana and J. C. Lagarias, *One-way functions and circuit complexity*, manuscript.

[Bo-77] A. Borodin, *On relating time and space to size and depth*, SIAM J. Comput. 6, 733-743.

[Bo-85] S. W. Boyack, *The robustness of combinatorial measures of Boolean matrix complexity*, Doctoral Dissertation, M.I.T.

[Br-77a] F.-J. Brandenburg, *On one-way auxiliary pushdown automata*, Proc. 3rd GI Conference, Lecture Notes in Computer Science 48, pp. 133-144.

[Br-77b] F.-J. Brandenburg, *The contextsensitivity of contextsensitive grammars and languages*, Proc. 4th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science 52, pp. 120-134.

[Br-80] F.-J Brandenburg, *Multiple equality sets and Post machines*, J. Computer and System Sciences 21, 292-316.

[Br-79] G. Brassard, *A note on the complexity of cryptography*, IEEE Transactions on Information Theory IT-25, 232-233.

[Br-83] G. Brassard, *Relativized cryptography*, IEEE Transactions on Information Theory IT-29, 877-894.

[CKS-81] A. K. Chandra, D. C. Kozen, L. J. Stockmeyer, *Alternation*, J.ACM 28, 114-133.

[CSV-84] A. K. Chandra, L. J. Stockmeyer, and U. Vishkin, *Constant depth reducibility*, SIAM J. Comput. 13, 423-439.

[Ch-77] M. P. Chytil, *Comparison of the active visiting and the crossing complexities*, Proc. 6th Conference on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science 53, pp. 272-281.

[CJ-77] M. P. Chytil and V. Jákl, *Serial composition of 2-way finite-state transducers and simple programs on strings*, Proc. 4th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science 52, pp. 135-147.

[Co-71] S. Cook, *Characterizations of pushdown machines in terms of time-bounded computers*, J. ACM 19, 175-183.

[Co-79] S. Cook, *Deterministic CFL's are accepted simultaneously in polynomial time and log squared space*, Proc. 11th Annual ACM Symposium on Theory of Computing, pp. 338-345.

[Co-81] S. A. Cook, *Towards a complexity theory of synchronous parallel computation*, L'Enseignement Mathematique 27, 99-124.

[Co-83] S. Cook, *The classification of problems which have fast parallel algorithms*, Proc. 4th International Conference on Foundations of Computation Theory, Lecture Notes in Computer Science 158, pp. 78-93.

[DKR-76] A. Demers, C. Kelemen, B. Reusch, *On encryption systems realized by finite transducers*, technical report TR 76-291, Cornell University.

[DKR-82] A. Demers, C. Kelemen, B. Reusch, *On some decidable properties of finite state translations*, Acta Informatica 17, 349-364.

[DDD-83] R. DeMillo, G. Davida, D. Dobkin, M. Harrison, and R. Lipton, Applied Cryptology, Cryptographic Protocols, and Computer Security Models, Proceedings of Symposia in Applied Mathematics, vol. 29, American Mathematical Society.

[DH-76] W. Diffie and M. E. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory IT-22, 644-654.

[Do-82] M. Dowd, *Isomorphism of complete sets*, Tech. Report LCSR-TR-34, Rutgers University.

[DC-80] P. W. Dymond and S. A. Cook, *Hardware complexity and parallel computation*, Proc. 21st IEEE Symposium on Foundations of Computer Science, pp. 360-372.

[FMR-68] P. Fischer, A. Meyer, and A. Rosenberg, *Counter machines and counter languages*, Mathematical Systems Theory 2, 265-283.

[Fl-66] M. Flynn, *Very high-speed computing systems*, Proceedings of the IEEE, 54, 1901-1909.

[Ga-77] Z. Galil, *Some open problems in the theory of computation as questions about two-way deterministic pushdown automaton languages*, Mathematical Systems Theory 10, 211-228.

[GP-83] Z. Galil and W. Paul, *An efficient general-purpose parallel computer*, J. ACM 30, 360-387.

[GJ-79] M. R. Garey and D. S. Johnson, Computers and Intractability, Freeman and Co., San Francisco.

[vzG-84] J. von zur Gathen, *Parallel powering*, Proc. 25th IEEE Symposium on Foundations of Computer Science, pp. 31-36.

[Gi-77] J. Gill, *Computational complexity of probabilistic Turing machines*, SIAM J. Comput. 6, 675-695.

[GGH-67] S. Ginsburg, S. Greibach, and M. Harrison, *Stack automata and compiling*, J. ACM 14, 172-201.

[GS-85] A. V. Goldberg and M. Sipser, *Compression and Ranking*, Proc. 17th Annual ACM Symposium on Theory of Computing, pp. 440-448.

[Go-82] L. M. Goldschlager, *A universal interconnection pattern for parallel computers*, J. ACM 29, 1073-1086.

[Gr-69] S. A. Greibach, *Checking automata and one-way stack languages*, J. Computer and System Sciences 3, 196-217.

[Gr-78a] S. A. Greibach, *Hierarchy theorems for two-way finite state transducers*, Acta Informatica 11, 89-101.

[Gr-78b] S. A. Greibach, *One way finite visit automata*, Theoretical Computer Science 6, 175-221.

[Gr-84] J. Grollmann, *Complexity measures for public-key cryptosystems*, Doctoral Dissertation, Universität Dortmund.

[GS-84] J. Grollmann and A. Selman, *Complexity measures for public-key cryptosystems*, Proc. 25th IEEE Symposium on Foundations of Computer Science, pp. 495-503.

[GI-81] E. M. Gurari and O. H. Ibarra, *The complexity of decision problems for finite-turn multicounter machines*, Proc. 8th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science 115, pp. 495-505.

[Gu-83] Y. Gurevich, *Algebras of feasible functions*, Proc. 24th IEEE Symposium on Foundations of Computer Science, pp. 210-214.

[Ha-78] J. Hartmanis, *On log-tape isomorphisms of complete sets*, Theoretical Computer Science 7, 273-286.

[Ha-83] J. Hartmanis, *Generalized Kolmogorov complexity and the structure of feasible computations*, Proc. 24th IEEE Symposium on Foundations of Computer Science, pp. 439-445.

[HIM-78] J. Hartmanis, N. Immerman, S. Mahaney, *One-way log-tape reductions*, Proc. 19th IEEE Symposium on Foundations of Computer Science, pp. 65-72.

[HM-80] J. Hartmanis, S. Mahaney, *An essay about research on sparse NP-complete sets*, Proc. 9th Conference on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science 88, pp. 40-57.

[HM-81] J. Hartmanis, S. Mahaney, *Languages simultaneously complete for one-way and two-way log-tape automata*, SIAM J. Comput. 10, 383-390.

[HY-84] J. Hartmanis and Y. Yesha, *Computation times of NP sets of different densities*, Theoretical Computer Science 34, 17-32.

[Ho-80] Jia-wei Hong, *On similarity and duality of computation*, Proc. 21st IEEE Symposium on Foundations of Computer Science, pp. 348-359.

[HU-67] J. Hopcroft and J. Ullman, *Nonerasing stack automata*, J. Computer and System Sciences 1, 166-186.

[HU-79] J. E. Hopcroft and J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, Mass.

[Ib-78] O. H. Ibarra, *Reversal-bounded multicounter machines and their decision problems*, J. ACM 25, 116-133.

[Ib-82] O. H. Ibarra, *2DST mappings on languages and related problems*, Theoretical Computer Science 19, 219-227.

[Ib-83] O. H. Ibarra, *On some decision questions concerning pushdown machines*, Theoretical Computer Science 24, 313-322.

[Jo-84] D. S. Johnson, *The NP-completeness column: an ongoing guide*, Journal of Algorithms 5, 284-299.

[Jo-75] N. D. Jones, *Space-bounded reducibility among combinatorial problems*, J. Computer and System Sciences 11, 68-85.

[JY-85] D. Joseph and P. Young, *Some remarks on witness functions for non-polynomial and non-complete sets in NP*, to appear in Theoretical Computer Science.

[KL-82] R. M. Karp and R. J. Lipton, *Turing machines that take advice*, L'Enseignement Mathematique 28, 191-209.

[Ki-81a] K. N. King, *Measures of parallelism in alternating computation trees*, Proc. 13th Annual ACM Symposium on the Theory of Computing, pp. 189-201.

[Ki-81b] K. N. King, *Alternating multihead finite automata*, Proc. 8th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science 115, pp. 506-520.

[Ko-83] K.-I. Ko, *On the definition of some complexity classes of real numbers*, Mathematical Systems Theory 16, 95-109.

[Ko-84] K.-I. Ko, *A definition of infinite pseudorandom sequences*, manuscript, University of Houston.

[Ko-81] A. Konheim, Cryptography, A Primer, Wiley, New York.

[Ku-83] S. A. Kurtz, *A relativized failure of the Berman-Hartmanis conjecture*, draft, Department of Mathematics, University of Chicago, May.

[Le-79] A. Lempel, *Cryptology in transition*, Computing Surveys 11, 285-303.

[Le-85] L. A. Levin, *One-way functions and pseudorandom generators*, Proc. 17th Annual ACM Symposium on Theory of Computing, pp. 363-365.

[Li-85] M. Li, *Lower bounds on string-matching*, manuscript.

[LL-78] N. Lynch and R. Lipton, *On structure preserving reductions*, SIAM J. Comput, 7, 119-126.

[Ma-69] G. Mager, *Writing pushdown acceptors*, J. Computer and System Sciences 3, 276-319.

[Ma-82] S. Mahaney, *Sparse complete sets for NP: Solution of a conjecture of Berman and Hartmanis*, J. Computer and System Sciences 25, 130-143.

[MY-85] S. Mahaney and P. Young, *Reductions among polynomial isomorphism types*, to appear in Theoretical Computer Science.

[Mi-76] G. L. Miller, *Riemann's hypothesis and tests for primality*, J. Computer and System Sciences 13, 300-317.

[MS-80] B. Monien and I. Sudborough, *Bandwidth problems in graphs*, Proc. 18th Annual Allerton Conference on Communication, Control, and Computing, pp. 650-659.

[MS-81] B. Monien and I. Sudborough, *Bandwidth constrained NP-complete problems*, Proc. 13th Annual ACM Symposium on Theory of Computing, pp. 207-217.

[MT-79] R. Morris and K. Thompson, *Password security: a case history*, CACM 22, 594-597.

[Pi-79] N. Pippenger, *On simultaneous resource bounds*, Proc. 20th IEEE Symposium on Foundations of Computer Science, pp. 307-311.

[Pi-81] N. Pippenger, *Pebbling with an auxiliary pushdown*, J. Computer and System Sciences 23, 151-165.

[PF-79] N. Pippenger and M. J. Fischer, *Relations among complexity measures*, J. ACM 26, 361-381.

[Ra-79] M. O. Rabin, *Digital signatures and public-key functions as intractable as factorization*, technical report MIT/LCS/TR-212, M.I.T.

[Ra-80] M. O. Rabin, *Probabilistic algorithm for testing primality*, J. Number Theory 12, 128-138.

[Ra-85] C. Rackoff, personal communication.

[Re-83] K. W. Regan, *On diagonalization methods and the structure of language classes*, Proc. 4th International Conference on Foundations of Computation Theory, Lecture Notes in Computer Science 158, pp. 368-380.

[Re-85] K. W. Regan, *A uniform reduction theorem with application to the complexity of inverting easy-to-compute functions*, manuscript.

[RT-84] J. H. Reif and J. D. Tygar, *Efficient parallel pseudo-random number generation*, technical report TR-07-84, Harvard University.

[RSA-78] R. L. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, CACM 21, 120-126.

[Ro-66] H. Rogers, Recursive Functions and Effective Computability, McGraw-Hill, New York.

[Ro-73] W. C. Rounds, *Complexity of recognition in intermediate-level languages*, IEEE Conference Record of the 14th Annual Symposium on Switching and Automata Theory, pp. 145-158.

[Ru-80] W. L. Ruzzo, *Tree-size bounded alternation*, J. Computer and System Sciences 21, 218-235.

[Ru-81] W. L. Ruzzo, *On uniform circuit complexity*, J. Computer and System Sciences 22, 365-383.

[Ru-85] W. L. Ruzzo, personal communication.

[Sa-72] J. E. Savage, *Computational work and time on finite machines*, J. ACM 19, 660-674.

[SY-82] A. Selman and Y. Yacobi, *The complexity of promise problems*, Proc. 9th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science 140, pp. 502-509.

[Sh-81] A. Shamir, *On the generation of cryptographically strong pseudo-random sequences*, Proc. 8th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science 115, pp. 544-550.

[Sh-49] C. E. Shannon, *Communication theory of secrecy systems*, Bell System Technical Journal 28, 656-715.

[Si-82] M. Sipser, *On relativization and the existence of complete sets*, Proc. 9th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science 140, pp. 523-531.

[SS-77] R. Solovay and V. Strassen, *A fast Monte-Carlo test for primality*, SIAM J. Comput. 6, 84-85.

[SHL-65] R. E. Stearns, J. Hartmanis, and P. M. Lewis II, *Hierarchies of memory limited computations*, IEEE Conference Record on Switching Circuit Theory and Logical Design, pp. 179-190.

[St-74] L. J. Stockmeyer, *The complexity of decision problems in automata theory and logic*, Doctoral Dissertation, M.I.T.

[Su-75a] I. H. Sudborough, *On tape-bounded complexity classes and multihead finite automata*, J. Computer and System Sciences 10, 62-76.

[Su-75b] I. H. Sudborough, *A note on tape-bounded complexity classes and linear context-free languages*, J. ACM 22, 499-500.

[Su-78] I. H. Sudborough, *On the tape complexity of deterministic context-free languages*, J. ACM 25, 405-414.

[Su-83] I. H. Sudborough, *Bandwidth constraints on problems complete for polynomial time*, Theoretical Computer Science 26, pp. 25-52.

[Va-76] L. Valiant, *Relative complexity of checking and evaluating*, Information Processing Letters 5, 20-23.

[Vi-83] U. Vishkin, *Synchronous parallel computation—a survey*, Preprint, Courant Institute, New York University.

[Vi-84] U. Vishkin, *A parallel-design distributed-implementation (PDDI) general-purpose computer*, Theoretical Computer Science 32, 157-172.

[Vi-81] P. M. B. Vitányi, *A note on DPDA transductions of {0,1}* and inverse DPDA transductions of the Dyck set*, Intern. J. Computer Math. Section A, Vol. 9, 131-137.

[VS-78] P. M. B. Vitányi and W. J. Savitch, *On inverse deterministic pushdown transductions*, J. Computer and System Sciences 16, 423-444.

[Wa-85] Osamu Watanabe, *On one-one polynomial time equivalence relations*, to appear in Theoretical Computer Science.

[We-79] G. Wechsung, *The oscillation complexity and a hierarchy of context-free languages*, Proc. 2nd International Conference on Fundamentals of Computation Theory, Akademie-Verlag, Berlin, GDR, pp. 508-515.

[We-80] G. Wechsung, *A note on the return complexity*, Elektronische Informationsverarbeitung und Kybernetik 16, 139-146.

[WB-79] G. Wechsung and A. Brandstädt, *A relation between space, return and dual return complexities*, Theoretical Computer Science 9, 127-140.

[Wi-68] M. V. Wilkes, Time-Sharing Computer Systems, MacDonald/American Elsevier, New York.

[Ya-82] A. C. Yao, *Theory and applications of trapdoor functions*, Proc. 23rd IEEE Symposium on Foundations of Computer Science, pp. 80-91.

[Yo-83] P. Young, *Some structural properties of polynomial reducibilities and sets in NP*, Proc. 15th Annual ACM Symposium on Theory of Computing, pp. 392-401.

[Za-82] S. Zachos, *Robustness of probabilistic computational complexity classes under defnitional perturbations*, Information and Control, 54 143-154.

VITA

Eric Warren Allender was born November 19, 1956 in Mt. Pleasant, Iowa. He graduated from Mt. Pleasant High School in 1975, and then attended the University of Iowa, receiving the B. A. degree with majors in Theatre and Computer Science in 1979.