Access Complexity

May 7, 2001

A thesis presented

for partial fulfillment of the requirements for

Honors in Mathematics

Bachelor of Arts

Cornell University

by

Ryan Williams

Abstract

We propose new formalisms for representing the fundamental complexity classes L, NL, P, NP, P^{NP}, and $PSPACE$. The idea is to start with a weak Turing machine model (we choose logspace TMs) and augment the model with an extra tape, which we will call the "reference." Models capturing various complexity classes arise from three conditions imposed on the reference:

(1) the length of the tape with respect to the input size,

(2) the access restrictions to the tape (i.e. how the tape may be read or written to),

(3) the initial contents of the tape with respect to an input.

We find many natural kinds of tape restrictions where the resulting model captures the classes above. For example, those machines such that the reference can be accessed two-way, read, and write-once accept exactly those sets in P.

Using this new formulation, we prove several new theorems relating classes, and generalize some of our results to more classes. We call the study of such models "access complexity", since the fundamental differences between the classes here are in the accessibility of massive storage.

Table of Contents

1 Acknowledgements

I would first like to recognize my research advisor Juris Hartmanis, who has listened patiently to all of my results with a watchful eye and a great deal of encouragement. Much of the synthesis prevailing throughout the text– the big picture commentary of where and why access complexity is an interesting and significant field of study– is the direct result of many enlightening discussions with Professor Hartmanis.

Among those who have given me priceless advice during my undergraduate study, I give many thanks to Anil Nerode, Charles Van Loan, Sergei Artemov, Dieter van Melkebeek, and David Gries. There are many others who are not listed. This is due not to their lack of good advice, but my lack of long-term memory.

For continuously helping me preserve my sanity throughout these four years, I am indebted to Kate Chabarek.

Finally, I wish to thank my parents, who with enormous understanding have allowed me to attend this wonderful institution.

And for no particular reason, I acknowledge my current roommate, John Briggs. Hey, buddy!

2 Introduction.

From its beginning, computational complexity has been focused on the study of how quantitatively measured resources affect computation– what can and cannot be decided when algorithms are allowed bounded resources which grow with the size of the input. The intuitive notions of nondeterminism [13] versus alternation [2], and time [5] versus space [6] as resources have been cornerstones in the development of the theory. Complexity theorists have placed a strong emphasis on understanding computable sets via the time and space requirements of deterministic, nondeterministic, or alternating machines recognizing them.

Such a philosophy has given rise to robust complexity classes of problems solvable by machines with some particular bounded resource. The complexity classes L, NL, P, NP, and $PSPACE$ have been by far the most studied of these classes, especially P and NP. We would like to know precisely how these classes are related; we know that each class in the list above is contained in its successor, and we know that $NL \neq PSPACE$, but this is the extent of our knowledge, more or less. For example, we cannot say if L is properly contained in NP, or if P is properly contained in $PSPACE$. It has been steadfastly conjectured that both of these propositions are true. But it appears that our current proof methods are woefully inadequate for resolving these problems.

In this work, we present a novel way to represent the major complexity classes. We homogenize the model of computation under which the classes are defined, and express the conjectured differences between the classes as differences between read-only vs. read/write-once vs. read/write capabilities, and 1-way head vs. 2-way head capabilities. That is, we express the conjectured differences between the classes in the form of *access permissions*. This is done by introducing to a logspace machine an extra "reference" tape of polynomial or exponential size, and placing various restrictions on how this tape may be accessed. The result is that we are able to express the complexity classes in terms of the *how massive storage is accessed* rather than how much time/space the machine requires. Under our canonical formulation, NL, P, NP, P^{NP}, and $PSPACE$ machines all have polynomial space available to them. *How they use the space* is what makes them different.

3 Preliminaries

We assume familiarity with Turing machines, basic hierarchy theorems, and the classes L, P, NP, P^{NP}, and $PSPACE$.[7] Our coverage here will not be overly technical; rather, we will precisely define what all of the pieces of our machines will do, taking it as a matter of course that such things could be easily rigorously defined. The significant workings of our model are all contained in their conceptual meanings.

3.1 Augmentations and references

Definition 3.1 *Let M be a Turing machine. An k* **head,** *X* **permission,** *$f(n)$-*
augmentation of *M is a machine M' which consists of M with k additional heads accessing an additional tape W of $f(n)$ size, where n is the length of the input to M. W is completely independent of the input, work, and output tapes of M, and will be called the* **reference** *of M'. The means by which the k heads can access W are dictated by the expression X; possible values for X will be given later.*

Our definition of an augmentation is intentionally vague with respect to accessing the reference. What will be computable with the augmentation M' depends upon the capability of the head that is reading and writing to W. Furthermore, we will allow either the initial contents of W to vary, or W will be forced to be initially blank.

By default, unless otherwise specified, we drop the "k heads" designation and assume that k is arbitrary in our statements; we will also do this similarly with X.

We first deal with the issues of what the reference will initially hold.

Definition 3.2 *(Initial contents of references)*

Let M' be an $f(n)$-augmentation of some M, and let W be the reference of M'. The **initial content of W on input** *x is defined as a string $y, |y| = f(n)$ to appear on W when M' is in its initial configuration on x. Note there may be more than one initial content.*

We say that W is **arbitrary content** *iff the initial content of W is dependent on the input x to M'. That is, for all inputs x, there exists a string y_x which is the initial contents of W.*

W is **blank** *iff for all inputs x, $y_x = B^{f(n)}$ in the above, where B is the blank symbol.*

It should be obvious that arbitrary content has some direct analogy with nondeterminism: for all inputs x, there exists a string y_x which is given to us for computation. Later on, we will see that access restrictions can dictate what kind of nondeterministic resources (space or time) our augmentations can simulate.

Just like determinism and nondeterminism, the two kinds of tapes have different acceptance criteria:

Definition 3.3 *Let M' be an X-permission $f(n)$-augmentation of M, and let W be the reference of M'.*

If W is blank, then $M'(x)$ **accepts** *if and only if $M(x)$ accepts when given W and access permissions X to W.*

If W is arbitrary content, $M'(x)$ **accepts** *iff there exists some string y_x such that when given y_x as the initial content of W, $M(x)$ accepts when given W and access permissions X to W.*

In other words, in the case where W is blank, acceptance is what you think it is, and when W is arbitrary content, acceptance occurs if there is something which can be initially written to W such that the augmentation accepts.

3.2 Access Permissions

In the previous section, we abstracted away the precise access permissions of our augmentations of M, without giving an example of what such an access permission would be like. We will now eliminate this difficulty by defining the range of possible access permissions we will consider in this work.

Access permissions are simply specifications on how the reference of an augmentation can be accessed. In the following definitions, let C be a space bounded complexity class. That is, for some space constructible f_C, $C = SPACE[f_C]$. We will say that a Turing machine M is a *C-machine* if $L(M) \in C$. Let $f(n)$ be space constructible in C.

Definition 3.4 *(1-Way Read-Only Accessibility)*

$C + 1f(n)RO$ is the class of languages accepted by 1-head, 1-way read-only permission, $f(n)$-augmented C-machines, with arbitrary content reference. That is, one head has 1-way, read-only access to the reference W: it moves in only one direction, and no symbols may be written to the tape.

Definition 3.5 *(Read-Only Accessibility)*

$C + f(n)RO$ is the class of languages accepted by 1-head, (2-way) read-only permission, $f(n)$-augmented C-machines with arbitrary content reference, i.e. for each C-machine M accepting a language in the class, the single head accessing the reference W is two-way read-only.

Definition 3.6 *(Blank, Write Once Accessibility)*

Define $C + f(n)WO$ is the class of languages accepted by 1-head, (2-way, read-only) write-once permission, $f(n)$-augmented C-machines with blank reference.

Definition 3.7 *(Full Accessibility)*

Define $C+f(n)RW$ is the class of languages accepted by 1-head, read-write permission, $f(n)$-augmented C-machines, i.e. the reference may be treated as an additional worktape of the C-machine.

These are the access permissions to be discussed in the first seven sections. In Section 8, we will add "increment accessibility" to this list.

Our convention will be that if $f(n)$ represents an arbitrary polynomial, we will denote it by P. Thus for example, $L + PRW$ is the class of languages accepted by 1-head reading and writing to polynomial-augmented logspace machines. This gives us a first, but trivial equivalence.

Corollary 3.1 $L + PRW = PSPACE$.

4 Main Characterizations

In our discussions, we will mostly restrict ourselves to the case where $C = L$.

4.1 NL

The following result relates restricted access to the reference with space-bounded non-determinism.

Theorem 4.1 $L + 1PRO = NL$.

Proof. We start with $L+1PRO \subseteq NL$. Let M be an augmented logspace machine with T of the required form. We construct a nondeterministic logspace machine M' such that $L(M) = L(M')$.

Begin $M'(x)$**:**

Guess the first symbol of T on $M(x)$ and write it to worktape location T'. Write the initial state of $M(x)$ on tape (for the input tape, store only the head position, so we use only logspace).

While the simulation of $M(x)$ hasn't halted, do:

(1) Execute a step of $M(x)$ using the current symbol at T' as the contents of T.

(2) If the head of T in M is supposed to move right during the step, then guess a symbol and write it to T', overwriting the previous symbol.

Accept $x \iff$ the simulation of $M(x)$ accepted.

End of $M(x)$.

Clearly, M' is an NL machine. Our claim is that the state of the simulated $M(x)$ in $M'(x)$ is σ iff there is a y such that the state $M(x)$ with $T := y$ is σ.

The proof is by induction on $|y|$. The claim clearly holds when y is the empty string, since then the head of T never moves, so $M'(x)$ is actually deterministic, in which case it behaves exactly like $M(x)$ by definition.

Suppose the claim is true for $|y| = n$. Let's assume there exists ya such that $M(x)$ when $T := ya$. Let the time at which the head of T reaches the square holding a be t. By induction hypothesis, after t steps of both machines, the simulated state of $M(x)$ in M' is equal to the state of $M(x)$.

For the remaining execution of $M(x)$, if the head of T does not move right to a then we are done. If the head does move right, then $M'(x)$ will just guess a and write it as the current symbol. In this case, $M'(x)$ accepts.

Conversely, if $M'(x)$ accepts, then that means it guessed an a such that from its state s at time t, it performed a deterministic simulation of $M(x)$ that accepted. By

induction there is a y such that from $M(x)$ at time t, $T := y$ will be in s. Since any deterministic parts of the two simulations yield the same result, $T := ya$ will yield $M(x)$ accepting.

$NL \subseteq L+1PRO$: Given an NL machine M, we assume WLOG that for any configuration, there are at ≤ 2 transitions applicable to that configuration. Furthermore, we assume an ordering on the transitions. We construct the following augmented M':

Begin $M'(x)$**:**

Initially, write the initial configuration of $M(x)$ on tape (head position of input tape, blank logspace worktape, initial state).

While the simulation of $M(x)$ hasn't halted:

(1) If there are 2 applicable transitions for the current configuration of $M(x)$, then read the current bit of T. If the bit is 0, choose the lesser of the two transitions (wrt to the ordering we assumed). Otherwise choose the greater of the two. Simulate $M(x)$ on the chosen transition and move the head of y to the right.

(2) Otherwise, execute $M(x)$ (deterministically) for 1 step.

Accept x iff the simulation accepted.

End of $M'(x)$

We claim that there is a y such that $M'(x)$ accepts when $T := y$ iff $M(x)$ accepts. The reasoning is as follows.

$M(x)$ accepts $\iff$ there is a sequence of transition "choices" such that executing $M(x)$ on those choices leads to acceptance

$\iff$ there is a bit string y specifying those choices, with its ith bit as 0 if the ith chosen transition is the lesser of the two and 1 otherwise, such that when $T := y$, the simulation of $M(x)$ in $M'(x)$ accepts, i.e. $M'(x)$ itself accepts.

□

A simple corollary that follows from the result is that 1-way read-only access of the reference augmented on NL machines does not increase the overall power of the machines. This is because the action of the head and the tape contents can be reproduced by a sequences of guesses by the NL machine. In other words,

Corollary 4.1 $NL + 1PRO = NL$.

In general, we have the following, which we will leave to the reader.

Theorem 4.2 *Let C, D be complexity classes, X be an accessibility requirement as given above, and $C + X = D$. Then $D + X = D$.*

4.1.1 NL using sublogarithmic space

A different characterization of NL can be found if we trade working storage for access permissions. In particular, if we let the read-only access to the reference be two-way instead of one-way, but restrict the Turing machine model to use only $O(\log \log n)$ space, the resulting complexity class is still NL.

Theorem 4.3 $NL = SPACE[\log \log n] + PRO$

Proof. (Sketch) Given an NL machine M, we construct a $SPACE[\log \log n] + PRO$ machine N that verifies valid computation histories (valcomps [7, 9]) on its reference using only $\log \log n$ space. We describe an N with this property below.

On input x, the contents of the polynomial reference of N contains (1) a "priming sequence" used for constructing $\log \log n$ worktape, and (2) a valcomp of the NL machine M. This priming sequence is the usual $b_0 \# b_1 \# \cdots \# b_n$, where b_i is the binary representation of integer i. Therefore, when a polynomial reference with arbitrary content is given to N, $\log \log n$ is indeed constructible by the machine, because given this sequence of binary numbers, precisely $\log \log n$ space can be marked on the worktape.

We assume that each the transition of the NL machine M is assigned a unique integer, stored in the control of N. Each configuration C specified in the valcomp of

M on the reference has the current worktape contents, and the index of the transition chosen to take by the NL machine. (The worktape head positions can be computed using $\log \log n$ space, so this is kept in N's own working storage.)

Essentially, N uses the reference to simulate having both $\log n$ worktape and nondeterminism, by:

(a) keeping track of the worktape heads of M,

(b) moving the input head exactly as M does,

(c) storing and changing the current state of M,

(d) choosing the transitions specified by the valcomp (via a constant length integer), and

(e) verifying that what should be written to the worktape of M appears on the reference.

Finally, N must verify that the rest of the worktape contents from one configuration to the next is identical, except for at most one change due to a transition. N does this by having two $O(\log \log n)$ space counters: we will call them *square* and *mover*. Initially $square = 1$, $mover = 0$. N records the first symbol σ_1 on the worktape of the current configuration, moves $k \cdot \log n$ steps backwards on the reference for some constant k (incrementing *mover* at each step), then checks that current symbol σ_1' and σ_1 are the same, moves forwards once (incrementing *square*), recording the current

symbol, moving $k \cdot \log n$ steps (decrementing *mover*), etc. When *square* is equal to the current worktape position of the simulation, we check the transition. When *square* $= k \log n$, N is done. When N reaches the end of the reference and the simulation of M is in an accept state, we accept.

In the other direction, we want to mimic the $O(\log \log n)$ space machine's two-way access using logspace and one-way. It suffices to find a way to simulate not only the "forwards" one-way movements of the tiny space machine, but also the "backwards" movements; that is, when the machine moves its reference head to the left. The crucial point is that the worktape of a $O(\log \log n)$ space machine will begin to repeat a configuration after taking $k \cdot \log n$ steps to the left on the reference, for some constant k (k will be the number of states in the machine, times the constant for the $O(\log \log n)$ space machine's worktape). It thus suffices for our $L + 1PRO$ machine simulating the $SPACE[\log \log n] + PRO$ machine to continuously save the previous $k \cdot \log n$ symbols it has read from the reference to worktape.

□

This tradeoff between space and access permissions is an interesting and instructive example of precisely how greater permissions can be used in lieu of extra space or time resources.

We remark here but do not prove that a similar characterization of L can be found

using write-once, read-only access, rather than arbitrary content read-only access.

4.2 P

Our next result relates logspace plus write-only access to polynomial reference to general polynomial time. It is quite interesting that a model defined in terms of space bounds and access permissions can capture a robust time complexity class naturally.

Theorem 4.4 $L + PWO = P$.

Proof. $L + PWO \subseteq P$: We show that for any $L + PWO$ machine M, there is a polynomial $p(n)$ such that after $p(n)$ steps of computation on inputs of size n, M will cycle if it has not already accepted or rejected. Thus each $L + PWO$ machine M can be fixed with a counter that forces M to reject after $p(n)$ steps, and the language accepted by M is not changed.

Let M be an $L + PWO$ machine, x be an input to M, and c be such that the number of configurations of $M(x)$ is bounded from above by $|x|^c$. Suppose that the reference of $M(x)$ is $|x|^k$ symbols long, so there are at most $|x|^k$ writes to the reference.

Let $w(i)$ be the index of the step of $M(x)$ in which the ith write to the reference occurs. For example, if the first write to the reference occurs in step 6 of $M(x)$'s execution, then $w(1) = 6$.

Let

$$s(|x|) = \max_{i \,:\, 2 \le i \le |x|^k} [w(i) - w(i-1)].$$

That is, $s(|x|)$ represents the longest interval of steps that $M(x)$ executes without writing to the reference. When $s(|x|) \geq |x|^c$, then $M(x)$ cycles since no writes occur; therefore the only component being modified in M is the logspace machine, which cycles after $|x|^c$ steps. Furthermore, if M cycles then the reference cannot change during the cycle (since M cannot erase the tape, as long a write occurs M cannot cycle) and the logspace machine must cycle.

Hence M accepts if and only if at most $|x|^k$ writes to the reference occur, with at most $s(|x|) \leq |x|^c$ steps occurring between these writes.

We conclude $M(x)$ accepts if and only if the running time of $M(x)$ is $\leq |x|^{k+c}$. Thus $L(M) \in P$.

$P \subseteq L + PWO$: One method to show this is to generate the entire computation history of the polynomial time machine on reference, and verify it is correct using logspace.

Another proof uses the fact that $HornSAT$ is P-complete with respect to logspace (many-one) reductions. [1] The canonical polynomial time algorithm for $HornSAT$

[1] When we say "reduction" without qualification, we will always mean "many-one reduction" unless otherwise stated.

can be performed easily in $L + PWO$, by letting the reference contain those variables which are set to true. Initially, all variables are false (the tape is blank). For every clause of the form $(\top \rightarrow x_i)$ in the formula, the index i is written to the reference. For every clause of the form $(x_{i_1} \wedge \cdots \wedge x_{i_k} \rightarrow x_j)$, if $i_1, \ldots, i_k$ all appear on the reference, then j is appended to the tape. This is done until all clauses have been satisfied in this way, or a contradiction is found. Therefore for any set $S \in P$, we use a logspace machine to reduce S to $HornSAT$ and then solve $HornSAT$ using the logspace machine and write-once reference. The theorem follows.

□

4.3 NP

The characterization for NP is perhaps the most straightforward of any we will prove. We shall use the following fact:

Fact 4.1 *(Cook) [3]*

For all $S \in NL$, there is a logspace reduction from S to 3CNF-SAT.

The below result exposes a fundamental difference between NL and NP, relating NP with augmented machines when the reference can be accessed 2-way read-only.

Theorem 4.5 *$L + PRO = NP$.*

Proof. $NP \subseteq L + PRO$: First, note that when given a satisfying assignment, $F \in 3CNF\text{-}SAT$ can be verified in L. This is because each clause is at most $O(\log n)$ bits long, so we can verify each clause individually as satisfying the given assignment, using only logspace for each one. Hence, an $L+PRO$ machine accepting $3CNF-SAT$ begins on input F with the reference containing the satisfying assignment for F, and the assignment is verified as satisfying using the logspace machine.

Since every language in NP can be logspace reduced to $3CNF\text{-}SAT$, every language in NP can be accepted via a logspace reduction followed by a $L + PRO$ computation (which is all in all an $L + PRO$ computation).

$L + PRO \subseteq NP$: Let M be an $L + PRO$ machine. The corresponding NP machine N on input x guesses the contents of the reference, then simulates the logspace machine of $M(x)$ with the guessed contents in polynomial time.

□

4.3.1 Using less than logspace

Access complexity is interesting because it unifies the complexity classes using one particular space-bounded model of computation. One may inquire as to how small this space-bounded model needs to be. As it stands, the difficulty between separating the models of access complexity lies in the power of logspace reductions, not in the

access mechanisms (which by themselves have varying levels of power). We would like to be able to replace the logspace machine with an even weaker model of computation. Here we demonstrate a result of this form for NP.

We define the complexity class $\Pi_3{-}TIME[\log n]$:

$S \in \Pi_3{-}TIME[\log n] \iff$

$S = \{x : (\forall w : |w| \leq \log |x|)(\exists y : |y| \leq \log |x|)(\forall z : |z| \leq \log |x|)\ R(x, w, y, z)\}$, where $R(\cdot, \cdot, \cdot\cdot)$ is a predicate that is solvable on a random-access machine running in time $O(\log n)$. (To recall, a random access machine is a Turing machine that on input x has an $O(\log |x|)$ length "index tape", upon which it may write an integer i. In the next step of the machine, the input tape head is defined to be situated at the ith input tape square.)

The result is that we can replace the logspace machine in our NP model with a logtime machine that uses only three alternations.

Theorem 4.6 $\Pi_3{-}TIME[\log n] + PRO = NP$.

Proof. (Sketch) One inclusion is obvious. To show than any NP machine can be modelled using a very small amount of computation, we give a predicate solvable in $\Pi_3{-}TIME[\log n]$ that expresses the acceptance of a valid computation history of an

NP machine on x. The fact that our polynomial reference can hold such a valid computation allows us to conclude the result.

For an NP machine N running in time n^c, let x be an input to the machine, and let all the configurations of $N(x)$ be of a fixed size $q \cdot n^d$ for some $d > c$. Furthermore, let us suppose that N has only one worktape, and q_0 be the initial state of N, and B be the blank symbol. Let h be a candidate for a valid computation history of $N(x)$ with this restriction.

Define $\texttt{LETTER}(x, i, n) = \sigma_1\sigma_2\cdots\sigma_n$ to be true if and only if the ith letter of x is σ_1, the $i+1$th letter is σ_2, ..., the $(i+n-1)$th letter is σ_n. Note that such equalities can be tested in constant time on a logtime machine, if $j - i$ is independent of the input size. (This definition was inspired by the work of Barrington, Immerman, and Straubing [1].)

It can be verified that $N(x)$ accepts if and only if:

(1) C_0 is valid (this can be checked in Π_1-$TIME[\log n]$),

(2) the final configuration C_{n^c} is accepting (this can be checked in Π_1-$TIME[\log n]$, assuming there is a unique accepting configuration), and

(3) for every configuration C_i, there exists a position k in C_i so that the following holds:

If the transition from C_i to C_{i+1} is $t = (q, \sigma) \to (q', \sigma', -1)$, then there is a $\rho \in \Sigma^*$

such that one of the following clauses $C(i,k,t)$ is true:

$L(\rho,i,k,t) = (\texttt{LETTER}(h, i\cdot|x|^c+k-1,3) = \rho q\sigma)\wedge(\texttt{LETTER}(h,(i+1)\cdot|x|^c+k-1,3) = q'\rho\sigma')$. (If t is not of the above form, we define $L(\rho,i,k,t)=0$.)

That is, for configuration i, if the head is in the kth position, then the transition t should write to the $(k+1)$th square, and move the head to the $(k-1)$th position.

For those transitions $t=(q,\sigma)\to(q',\sigma',+1)$, we define the following clause:

$R(i,k,t) = (\texttt{LETTER}(h, i\cdot|x|^c+k,2) = q\sigma)\wedge(\texttt{LETTER}(h,(i+1)\cdot|x|^c+k,2) = \sigma' q')$. (If t is not of the above form, we define $R(i,k,t)=0$.)

That is, for configuration i, if the head is in the kth position, then the transition taken should write to the kth square, and move the head to the $(k+1)$th position.

We also have predicates $P_L(j)$, $P_R(j)$ which checks those positions $j\notin\{k,k+1,k-1\}$ (all other letters appearing in the two configurations) are exactly the same. Depending on whether or not t moves the head to the left or right, the $(k-1)$th letter may or may not need to be checked. If the clause is of the form $L(i,k,t)$ ($R(i,k,t)$), the $P_L(j)$ ($P_R(j)$) predicate is used. This checking of equality of letters can be performed in Π_1-$TIME[\log n]$.

Formally, $N(x)$ accepts $\iff$

$(\forall 1\le i\le|x|^c)(\exists 1\le k\le|x|^c)$

$$\left[\left(\bigvee_{t\in T}[\vee_{\rho\in\Sigma}L(\rho,i,k,t)]\right)\wedge(\forall 1\leq j\leq|x|^c)[P_L(j)]\right]\vee\left[\left(\bigvee_{t\in T}R(i,k,t)\right)\wedge(\forall 1\leq j\leq|x|^c)[P_R(j)]\right]$$

which can be rewritten as:

$(\forall 1\leq i\leq|x|^c)(\exists 1\leq k\leq|x|^c)(\forall 1\leq j\leq|x|^c)$

$$\left[\left(\bigvee_{t\in T}[\vee_{\rho\in\Sigma}L(\rho,i,k,t)]\right)\wedge P_L(j)\right]\vee\left[\left(\bigvee_{t\in T}R(i,k,t)\right)\wedge P_R(j)\right]$$

The multiplication of various i values (of size $O(\log n)$) with n^c (written on tape as $1^{c\log n}$) can be done in Π_1-$TIME[\log n]$. Adding two i and j values together takes $O(\log n)$ time. Therefore, since the innermost quantifier is universal, we conclude that the entire process takes only three alternations; that is, it can be accomplished in $\Pi_3{-}TIME[\log n]$.

□

This tight characterization of NP is a surprise, since it appears that we cannot replace the logspace machine with a lesser machine in any of the other characterizations given (NL as $L+1PRO$, P as $L+PWO$ in the next subsection here, $PSPACE$ as $L+PI$ in Section 8, or $L+ERO$ in Section 5.2, P^{NP} as $L+PIR$ in Section 8). In all of these, replacing the logspace machine with a constant alternation logtime machine appears to weaken the machine significantly (especially in the $L+ERO$ case, where

such a machine cannot even specify a negligible fraction of squares on the reference). However, an intuition is never a final judgement, and due to the existence of complete problems using extremely restrictive reductions, such as first-order projections (see Immerman and Landau [12]), we are confident that weaker models can be used in lieu of logspace for some of these equivalences. These weaker models will allow us to focus more on the access permissions in our study of how complexity classes relate to one another.

4.4 Observations on the models

Contemplating on this new formulation, we may compare NL and NP in a simple yet enlightening way. NP is the set of problems that, if there is a short proof, then given this proof you may examine it all you like, reading backwards and forwards over it, and after polynomial time has passed, you either accept it as a valid proof or you don't. NL is the set of problems that, if there is a short proof, in your verification of its correctness you may only read the proof in one direction– there is no looking back at previous steps. When you are done with your one-way reading of the proof, you either accept it or reject.

Now we have reached a significant point in our reflection on NL versus NP. The conjectured difference between the two classes can be seen not only as an intuitive

separation of lesser and greater space/time resources, but as a difference in how the short proofs are allowed to be accessed. If a problem requires proofs so complex that any logspace computational device *must* peruse the proof as it verifies, introspectively looking back at previous parts of it to remind itself of bits/assignments/ formulas, etc., then this problem cannot be solved in NL. NL is only equipped for problems where the proof can be formulated so that the reader of the proof never has to look back at (more than a logarithmic number of) previous bits to verify. In a logical system, such a proof would have to resemble a linear chain of implications in order to have this property. (This is informal justification for why 2SAT is in NL; its proofs consist of linear chains of literals implying literals.)

Extending this analogy, one could say that P represents those problems that don't require a proof at all, but a logspace device can construct one given a blank reference. The device can examine its work all it wants; however it can never change its mind about the current proof it is writing. That is, once a bit of the proof has been written, it is cemented, and cannot be overwritten. This restriction on tape accessibility prevents the device from searching over an exponential number of possible proofs.

Furthermore, $PSPACE$ is those problems where, given a proof or not, one can overwrite pieces of the proof and read all over the proof, as much as one wants. One can "revise" the proof as much as one wants, verifying every possible one if necessary.

It is the class with no restrictions on the accessibility of the proof. In the next section, we will see a sharper characterization of $PSPACE$ which further elucidates the differences between this class and NP.

From this angle, it can almost seem strange that $NL \subseteq P$. On the left hand side, one is given a proof, and is allowed to read it in one direction; on the right hand side, one is given no proof at all, and is forced to construct one from scratch. Of course, one may reply that this situation just reveals the power of a two-way head on the reference: since it is exponentially larger than the worktape, being able to use it for storage and referencing old information is a huge advantage, even if the information constructed was done deterministically.

5 Relations with Finite Automata.

In this section, we use reductions from finite automata to logspace machines in order to find more access complexity characterizations of various classes, giving reproofs of the completeness of several finite automata problems along the way. We will see that the proposed models have a very strong and natural connection with various brands of finite automata.

5.1 DFA emptiness problems and access complexity

Our first result here shows what will be a general theme: the non-emptiness problem for DFAs with some kind of access to its input of a particular length is complete for $L + X$, where X denotes the same kind of access/length.

Theorem 5.1 *1DFAempty* $= \{\langle M\rangle \mid M$ *is a 1DFA and* $L(M) = \varnothing\}$ *is* NL*-complete with respect to* AC^0 *reductions.*

Proof. First, note that $\{\langle M, x, 1^{|x|^k}\rangle | M(x)$ is an $L + 1PRO$ machine that rejects x in $\leq |x|^k$ steps$\}$ is NL-complete (since $NL = coNL = L + 1PRO$) [11], [17].

The AC^0 circuit reducing $\langle M, x, 1^{|x|^k}\rangle$ to a DFA is as follows. Each possible configuration of the logspace machine in $M(x)$ will be treated as a state in the DFA.

That is, the state q, the $|x|$ symbols of the input tape, and the $\log |x|$ symbols of the worktape, and the input/worktape head positions all make up a single state of the DFA.

The AC^0 circuit deduces every possible transition from each configuration C of $M(x)$ to another configuration D. Each such transition is of the form $(q, i, \sigma, t) \rightarrow (q', \sigma', d, d')$, where i is the input symbol, σ is the worktape symbol, t is the symbol on the reference, q' is the new state, σ' is the symbol written, d is the direction for the input head, and d' is the direction for the worktape.

For a configuration C and a reference symbol t, there is a unique configuration D to which the logspace machine changes. For each pair of configurations, whether or not there is a transition from them when reference symbol t has been read can be computed separately in parallel, using only constant depth. Each transition will be thought of as a transition in the DFA that occurs on reading the reference symbol t. The initial configuration of the logspace machine is the start state, and the final states are any accepting configurations.

Now, $M(x)$ rejected iff the DFA N constructed accepts nothing.

□

In contrast, we give the following result for logarithmic space:

Theorem 5.2 *1DFAacc* $= \{\langle M, x\rangle \mid M$ *is a DFA and* $M(x)$ *accepts*$\}$ *is L-complete*

with respect to AC^0 reductions.

Proof. Reduction from the universal complete language $\{\langle M, x, 1^{|x|^k}\rangle | M$ is a logspace machine that rejects x in $\leq |x|^k$ steps$\}$. Use the AC^0 reduction in the previous proof, but instead of labeling the DFA transitions using the reference's symbols, use the symbols of x. In the reduction, carry over the same x in $\langle M, x, 1^{|x|^k}\rangle$ to the resulting $\langle M', x\rangle$.

□

Note that we can also use this reduction to show unary(1DFAempty) $= \{\langle M\rangle \mid M$ is a 1DFA with $\Sigma = \{0\}$ and $L(M) = \varnothing\}$ is L-complete. This result shows that $L + 1PRO$ where the reference alphabet of the machines are unary is equal to L.

Theorem 5.3 *2DFAnotempty$^{\leq n}$ $= \{\langle M\rangle \mid M$ is a 2DFA and $\exists x, |x| \leq |M|$ such that $x \in L(M)\}$ is NP-complete.*

Proof. The proof for 2DFAnotempty$^{\leq n}$ is analogous to the above proof, using 2DFAs with NP machines and the result that $L + PRO = NP$ (Theorem 4.4).

□

5.2 A $PSPACE$ characterization

Using these standard reductions from DFAs to logspace machines and from logspace machines to DFAs, it is very interesting to contrast the above two results with:

Theorem 5.4 *(Hunt, [8])*

2DFAnotempty $= \{\langle M \rangle \mid M$ *is a 2DFA and* $\exists x$ *such that* $x \in L(M)\}$ *is* $PSPACE$*-complete.*

This old result and the reductions between logspace and DFAs yield a different characterization of PSPACE. Let $L + ERO$ be the class of sets accepted by *doubly exponentially augmented logspace Turing machines*; that is, on inputs x the reference is of length $d^{|x|^k}$ for some $d > 1$, $k \geq 1$.

Theorem 5.5 $L + ERO = PSPACE$.

Proof. Use that: (1) 2DFAnotempty is $PSPACE$-complete, (2) there is a reduction given in previous proofs from augmented logspace Turing machines to 1DFAs and 2DFAs, and (3) if a 2DFA accepts any string at all, then it accepts a string of length exponential in the number of states.

□

The previous characterizations give us yet another interesting insight to an open problem: NP vs. $PSPACE$. NP is equivalent to logspace computation with two-way access to a polynomially long reference. From the above, $PSPACE$ is equivalent to logspace computation with two-way access to an arbitrarily long reference (but we can restrict it to exponential without any loss of generality). It is strange that we have a classification of $PSPACE$ which uses not polynomial space, but both logarithmic and exponential space! The logarithmic space can be manipulated in the usual way, while the exponential space is given nondeterministically yet it can only be read. Obviously, the logspace machine reading this exponential space will cycle many times while reading it, but the fact that the machine may move bidirectionally on the tape makes the problem non-trivial. The intuitive analogy that leads to our conjecture of $NP \neq PSPACE$ is that when computing devices are given proofs of a size that its memory can handle (that is, it does not cycle while reading in one direction), then what it is capable of recognizing is strictly less than when the computing device is given proofs of an enormous length. In the latter case, in order for the logspace machine to know its position on the reference, it is necessary for the reference to provide that information as well. (What is most interesting is that this is *all* the information the logspace machine needs in order to accept $PSPACE$, as we will see in the section on increment access.)

Translating the result that $NL \neq PSPACE$ into the language of our models is somewhat intriguing, and investigating what it means (the easiness of one-way with respect to two-way access in determining emptiness) in light of access complexity may well lead to new proof techniques. From this result we know that L with one-way read-only access to polynomial space is strictly less powerful than L with two-way read-only access to exponential space. However, why is it so difficult to show that $NL \neq NP$, or $NP \neq PSPACE$? This point is especially emphasized in the second case, where the space increase from one model to the other is exponential.

For another characterization of $PSPACE$, we may consider the set

$$\text{1DFAint} = \{\langle M_1, \ldots, M_n\rangle | M_i \text{ are } 1DFA, \textstyle\bigcap_{i=1}^{n} L(M_i) \neq \varnothing\},$$

which is $PSPACE$-complete [10]. Using this fact allows us to construct yet another machine model with computational power equivalent to $PSPACE$: a PSPACE machine can be denoted by an infinite family of $L+1PRO$ machines $(L_{i_1}, \ldots, L_{i_n}, \ldots)$ such that on inputs of size n, the first n machines in the family execute simultaneously on the input, and the overall machine accepts if they all accept and have the same contents on their reference. It is not difficult to show that there exists a machine of this type that can simulate arbitrary $L + 1PRO$ computations when given them as input.

Similarly, we can do the same for families of NP machines, yielding a result

concerning 2DFAint: any class that can accept 2DFAint is different from NP.

6 The "Repeated Proof" Method and NL vs. NP

We have suggested earlier in our more philosophical paragraphs that the new models for NL and NP may be useful in finding novel ways to relate the two classes, exploiting the intuitive disparity between one-way and two-way head movement. In this section, we give two examples of a technique that is used to relate one-way and two-way head movement in a unifying manner.

The gist of our idea is this: suppose we wish to solve an NP language $\mathcal{L}$ with an $L + 1PRO$ machine M'. In order to mimic two-way head movement with a unidirectional head, M' will do two things on an input x:

(1) Let y be a certificate that an $L+PRO$ machine N' accepting $\mathcal{L}$ would use to prove $x \in \mathcal{L}$. On the reference of M', we assume y is repeated every time the reference head of N' is supposed to move backwards. (The number of times this happens should be a polynomial, so the reference is still of polynomial length.) M' simulates N' using this information.

(2) Use another mechanism of M' to verify that the reference consists of some y being repeated.

6.1 An Upward Collapse Theorem

Via the aid of the NL and NP formulation, we show that NL with a one-way input head is extremely weak; weaker than a low level of the logarithmic hierarchy, unless $NL = NP$.

Similar to Section 4.3.1, we define Π_1-$TIME[\log n]$ by the following:

$$S \in \Pi_1\text{-}TIME[\log n] \iff S = \{x : (\forall y : |y| \leq \log |x|)\ R(x, y)\},$$

where $R(\cdot, \cdot)$ is a predicate that is solvable on a random-access machine running in time $O(\log n)$. (A reminder: a random access machine has some $O(\log n)$ length index tape, upon which it writes an integer i. In the next step, the input tape head is situated at the ith tape square.)

This class is extremely restrictive; it basically amounts to doing a polynomial number of $O(\log n)$ time computations in parallel, and accepting if and only if each computation branch accepts. Still, it is terribly unlikely that this class can be simulated by one-way nondeterministic logspace.

Theorem 6.1 *If* Π_1-$TIME[\log n] \subseteq$ *1-*NL*, then* $NL = NP$.

Proof. Let $L' = \{\langle x, n\rangle | \exists w \in \Sigma^*,\ \exists m \in N,\ [|w| = n, x = w^m]\}$. L' can be expressed in the following way:

$L' = \{\langle x, n\rangle | \forall i : |i| \leq (\log(|x| - n)), \texttt{BIT}(x, i) = \texttt{BIT}(x, i + n)\}$, where $\texttt{BIT}(x, j)$ is equal to the jth bit of string x. It is easy to see that the BIT predicate can be computed in $O(\log n)$; thus L' is in Π_1-$TIME[\log n]$. (BIT is defined in Barrington, Immerman, and Straubing [1].)

We show that if L is decidable in 1-way NL, then 3CNF SAT can be solved in NL. Suppose a 1-way NL-machine N accepts L. Then an $L + 1PRO$ machine deciding $3CNF$-SAT is the following.

On input formula F with n variables and m clauses, the machine has three stages which are interleaved:

(1) This stage has a 1-way read-only reference containing one variable assignment for F, written m times in succession (one copy of the assignment for each clause). We follow each of these copies with a special marker, so there are $n(m + 1)$ symbols on the tape. In this stage, the machine checks each clause of F individually with the current satisfying assignment it is reading from the reference. This can easily be done in logspace, even when the assignment is read in only one direction. This stage accepts iff every clause is satisfied by its corresponding assignment on the reference.

(2) In this stage, the machine verifies that the contents of the reference is indeed a string being repeated m times. This is done using the 1-way NL machine N, which considers the reference as an input tape, and using the reference head, determines

that it is a string being repeated by assumption. This stage accepts iff N accepts.

(3) A counter independently checks that the special marker on the reference is being guessed every $n+1$ steps (and in no other steps), to ensure that the string being repeated is indeed $n+1$ bits long.

The NL machine accepts $F \iff$ All three stages accept F

$\iff$ There is a truth assignment that satisfies all m clauses in F

$\iff F \in 3CNF\text{-}SAT.$

□

We note as a matter of course that this theorem could have been proven without the characterization of $L{+}1PRO$; however, we argue that the formalism chosen makes it vastly easier to visualize the satisfying assignment being "guessed repeatedly".

6.2 Two One-Way Heads are Powerful

Using the repeated proof method, it is also possible to show that the power of logspace machines given access to a reference via *two* one-way read-only heads is equivalent to that of NP. We define the class of languages accepted by such machines $L{+}2[1PRO]$. The proof of this theorem is sketched below.

Theorem 6.2 $L + 2[1PRO] = NP.$

Proof. It is easy to see that $L + 2[1PRO] \subseteq L + PRO$, since a two-way head can mimic two heads by storing the head positions of both on the working storage of the logspace machine.

For the other inclusion, we construct a machine recognizing $3SAT$, in which the reference contains the same certificate given in the previous proof: some satisfying assignment for the input formula is repeated m times, where m is the number of clauses in the input formula. We read each clause of the input formula one at a time, storing the literals of the current clause in working storage (they require only logspace).

The responsibility of the first head is to check that the current assignment it is reading is consistent with the current clause it is reading. The second head checks that the next assignment on the tape is identical to the one the first head. If either of the two checks fail, then the machine rejects, and if they both succeed, it accepts. The checking for the second head is done by situating the second head n tape squares to the right of the first one, by letting the second head move right n times initially. After that, the second head is moved right exactly when the first head is. The logspace machine matches the symbol under the second head with the symbol under the first head; these symbols should be the same until the last assignment is reached.

□

7 Strengthening Savitch's Theorem

7.1 The Logspace Case

In this section, we use the equivalence of reading 1-way arbitrary content tape with nondeterministic space, and provide a stronger version of Savitch's theorem [15], which states that for any space constructible $S(n) \geq \log n$, $NSPACE[S] \subseteq SPACE[S^2]$. By giving a closer inspection of the algorithm used in Savitch's theorem, it is an easy corollary of the theorem that $NTISP[T, S] \subseteq SPACE[S \cdot \log T]$, where $NTISP[T, S]$ is the class of languages accepted by Turing machines running in time $T(n)$ and space $S(n)$. Our theorem is another level of precision. We show that a simple modification of the nondeterministic machine under consideration allows us to conclude that $NTISP[T, S] \subseteq SPACE[S \cdot \log T']$, where $T'(n)$ is the number of *nondeterministic* steps taken during the computation; that is, the number of steps in the computation for which the transition function is not well-defined.

Although the following is covered by our previous definitions, we state it again for convenience:

Definition 7.1 *Let $f(n)$ be space constructible in L. Then $L+1f(n)RO$ is defined as the class of languages accepted by logspace Turing machines $f(n)$-augmented, arbitrary content reference, with 1-way, read-only access.*

Theorem 7.1 *(Savitch Generalization, logspace version)*

Let $f(n) \geq \log n$ be space constructible. Then $L + 1f(n)RO \subseteq SPACE[\log f(n) \cdot \log n]$.

Proof. If for all $c > 0$, $f(n) > n^c$ infinitely often, then the theorem follows easily since $L+1f(n)RO = NL$ for any superpolynomial $f(n)$, and $NL \subseteq SPACE[\log^2 n] \subsetneq SPACE[\log f(n) \cdot \log n]$.

Assume then that $\log n \leq f(n) \leq n^c$ for some $c > 0$. Our goal is to simulate a logspace machine M with 1-way read-only access to an $f(n)$ length reference using a small amount of space. Recall from the proof of $L+1PRO = NL$ that if the logspace machine M has 1-way read-only access to an $f(n)$ length reference, this is equivalent to saying that at most $f(|x|)$ of the steps taken during the computation of the augmented machine M' are nondeterministic steps. We will call *critical configurations* those configurations C of M such that when C steps to a configuration D, the reference head moves right. These configurations correspond precisely to the points in the computation where a nondeterministic step happens.

Let us enforce these nondeterministic steps to be "uniformly spaced" throughout the computation of $M(x)$; i.e. we modify M so that it is in a critical configuration exactly once every k steps of the computation, for some k. In particular, we will modify M so that a critical configuration of $M(x)$ is reached once every $T(|x|)$ deter-

ministic steps, where $T(n)$ is the running time of M. Our modified machine M^* will, on input x, have an extra worktape of length $\log T(|x|)$ acting as a counter initialized at 0. Since $f(n) \leq n^c$, and our logspace machine has $\leq n^d$ configurations for some $d > 0$, $T(|x|)$ is a polynomial. Hence this counter requires $O(\log n)$ space.

Each step of $M^*(x)$ simulates $M(x)$ for one step and increments the counter by one. M^* simulates $M(x)$ until $M(x)$ reaches a critical configuration C. (To detect that C is a critical configuration, we determine if the number of possible transitions out of C is greater than one. This can be done in $O(1)$ time.) Then, the simulation of $M(x)$ is "stalled" at C until the counter reaches $T(|x|)$. When this occurs, the counter resets to zero, the nondeterministic step is made, and the process begins again.

Let us call this modified machine M^*. Observe that the running time of M^* is $O(f(n) \cdot T(n))$. That is, after every $T(n)$ steps, a new square on the reference is read, and there are only $f(n)$ of these squares. The space usage of M^* is $O(\log n + \log T(n)) = O(\log n)$.

We now apply the traditional divide-and-conquer technique of Savitch's theorem to M^*, obtaining a machine M^{**} which simulates M^* and uses $O(\log n \cdot \log f(n))$ space. We consider a configuration C of $M(x)$ to be the worktape contents, input head position, and reference head position, so a configuration requires $O(\log n)$ space.

$M^{**}(x)$:

Let I be the initial configuration of $M^*(x)$.

Let A be the unique accepting configuration of $M^*(x)$.

Execute $REACH(I, A, f(|x|))$, and *accept* iff it accepts.

The algorithm $REACH(C, E, s)$ will accept iff there is a path from C to E that has $\leq s$ nondeterministic steps.

$REACH(C, E, s)$:

Execute M^* starting from C, until a critical configuration C' is reached.

If $(C' = E)$ then *accept*.

If $(s = 0)$, then *reject*.

For all configurations D of $M^*(x)$,

If $REACH(C', D, \lfloor s/2 \rfloor)$ and $REACH(D, E, \lceil s/2 \rceil)$ accept, then *accept*.

End for.

Reject.

The crucial observations to make in the above algorithm are that:

(a) We need not store the counter along with our configurations. For every recursive call of $REACH$, it is assumed that in configurations C and E, the counter has just been reset (i.e. set to 0). Note that this is preserved by our construction.

(b) We only make a recursive call when the reference is to be accessed. Two points: (1) This occurs $f(|x|)$ times during the computation, and (2) for configurations C, E that are input, if they are separated by $\leq T(|x|)$ steps, then no recursive call is made. This follows from (a).

(c) From the "uniformly spaced" construction of M^*, we are guaranteed that every time the two recursive calls of the procedure are made, both calls will encounter the same number of critical configurations (within one).

Thus, from (b) and (c) we conclude that at any point we will have at most $O(\log f(n))$ configurations saved to tape, and from (a) we conclude this corresponds to $O(\log f(n) \cdot \log n)$ space used to store configurations. We need $O(\log n + \log T(n))$ space to simulate the logspace machine, but as $T(n) \leq n^d$, this quantity is negligible.

□

Corollary 7.1 *For all* $k > 0$, $L + 1\log^k(n)RO \subseteq SPACE[\log(\log^k n) \cdot \log n] \subsetneq SPACE[\log^2 n]$.

Proof. Let $f(n) = \log^k n$ in the previous theorem, and apply the space hierarchy theorem. □

This shows that logspace with 1-way read-only polylog reference is properly contained in $O(\log^2 n)$ space, demonstrating further the power of reading and writing to

some proof versus reading some proof sequentially.

We do not know if $L + 1PRO = NL \subsetneq SPACE[\log^2 n]$, i.e. if logspace with polynomial 1-way reading is weaker than $\log^2 n$ space, but the above result establishes that logspace with sublinear 1-way reading is indeed weaker.

More formally,

Theorem 7.2 *If for some $c > 0$, $f(n) = o(n)^c$, then $L+1f(n)RO \subsetneq SPACE[\log^2 n]$.*

This is because $L + 1f(n)RO \subseteq SPACE[\log n \cdot c \cdot \log o(n)]$. In other words, logspace with one-way read-only sublinear advice is not as powerful as log-squared space. Hence we have a result that is very close to $NL \subsetneq SPACE[\log^2 n]$, but not quite. It is still an interesting result: the class we have specified clearly contains L, is most likely more powerful than L, and its corresponding machines run in polynomial time. However, it is properly contained in $SPACE[\log^2 n]$.

7.2 The General Case

Let $SPACEN[S(n), f(n)]$ be the class of languages accepted by $S(n)$ space machines taking at most $f(n)$ nondeterministic steps on inputs of size n, i.e. possessing a 1-way read-only reference of size $f(n)$. (Of course, $f(n)$ must be $S(n)$ space constructible.) Using the techniques of the $L + 1PRO = NP$ result (Theorem 4.4), it is easy to

see that $SPACEN[S(n), f(n)] = SPACE[S(n)] + 1f(n)RO$. We note some simple observations, all of which are easy to prove using standard simulation techniques:

(1) If $S_1(n) \in O(S_2(n))$ and $f_1(n) \in O(f_2(n))$ then $SPACEN[S_1(n), f_1(n)] \subseteq SPACEN[S_2(n), f_2(n)]$.

(1) If $S_1(n) \in o(S_2(n))$ and $f_1(n) \in o(f_2(n))$ then $SPACEN[S_1(n), f_1(n)] \subsetneq SPACEN[S_2(n), f_2(n)]$.

(3) $SPACEN[S(n), 2^{O(S(n))}] = NSPACE[S(n)]$.

Our result here is the expected one:

Theorem 7.3 *(Generalized Savitch)*

$SPACEN[S(n), f(n)] \subseteq SPACE[S(n) \log f(n)]$.

We use an argument analogous to the previous theorem, substituting $\log n$ for $S(n)$. The argument works because $T(n)$, the running time of the $S(n)$ space machine, is still exponential in $S(n)$ (so a $\log T(n)$ counter can be written to $S(n)$).

□

Our next result shows a tight relation between $SPACEN[S(n), o(2^{O(S(n))})]$ and $NSPACE[S(n)]$:

Lemma 7.1 $NSPACE[o(S(n))] \subseteq SPACEN[S(n), 2^{O(1)\cdot o(S(n)))}] \subseteq NSPACE[S(n)]$.

Proof. $NSPACE[o(S(n))] = SPACEN[o(S(n)), 2^{O(1)\cdot o(S(n))}]$

$\subseteq SPACEN[S(n), 2^{O(1)\cdot o(S(n))}]$.

□

So this class is sandwiched tightly between $NSPACE[S(n)]$ and all the classes below it. It is easy to see as a corollary that:

$$\text{Either } NSPACE[o(S(n))] = SPACEN[S(n), 2^{O(1)\cdot o(S(n)))}],$$

$$\text{or } SPACEN[S(n), 2^{O(1)\cdot o(S(n)))}] = NSPACE[S(n)].$$

Therefore, by considering one-way read-only access, we obtain interesting complexity classes that are "between" two classes which should not have any classes strictly between them. The upshot is a surprising one: that either $o(S(n))$ nondeterministic space can capture $S(n)$ deterministic space (with asymptotically fewer nondeterministic steps than the general case), or $S(n)$ nondeterministic space can be captured by taking asymptotically fewer nondeterministic steps than the general case.

The obvious open question here asks which is more powerful: asymptotically more worktape, or asymptotically more one-way read-only reference. Our intuition (and the models we have provided) tells us that the first is more powerful, though a proof that $SPACEN[S(n), 2^{O(1)\cdot o(S(n)))}] = NSPACE[S(n)]$ is not known.

8 The Complexity of Brute Force Search

The models we have provided have been quite natural in terms of access permissions. However, of the models we have considered, there is a large gap in computational power when the tape is initially blank: logspace and write-once on polytape is P, and writing arbitrarily many times is $PSPACE$. It would be nice to find a model which has initially blank tape, but captures complexity classes between P and $PSPACE$. This model would probably have the potential to run in exponential time, but its writes to the reference are still restricted in some sense.

In this vein, we formalize a model which represents a precise machine-theoretic characterization of the Russian notion of *perebor*, or an exhaustive "brute-force" search for a solution through all possible answers.[2] The model consists of a weak uninformed solution-checker, which simply tests every possible solution in lexicographic order, starting with the smallest in the ordering.

A natural question to ask of this model is how much information should be carried from the checking of one solution to the next. Should these processes be somewhat independent, in that no information is truly carried from one solution check to the next (other than the fact that all of the solutions lexicographically smaller than the current one have been examined thus far)? Or should we allow a small amount of

[2]For a highly informative survey of this work, see Trakhtenbrot's historical account [18].

information $O(\log n)$ to pass from one checking phase to the next? As expected, this measure directly determines the power of the model.

We propose the following model: a logspace machine with read-only access to a polynomial reference that holds a binary string which is initially all zeroes. The three distinctive sets of states for this machine will be the usual accept and reject states, and additional **increment states**. When a machine enters such a state, the binary number on the reference is incremented by one. Once the number is all ones, the computation is required to halt in either an accept or reject state. Thus the kind of access to the reference is twofold: reading, and incrementing. We will call such access to the reference **increment access**, for obvious reasons.

Intuitively, one would believe with little hesitation that SAT and perhaps also some lower levels of the polynomial time hierarchy can be solved using such a model. As it turns out, this model is quite powerful: it captures precisely $PSPACE$. We have established yet another equivalence between an access complexity based model and a traditional complexity class.

The arguments proving this are both simple and subtle. The implications of the equivalence are surprising. For example, all problems solvable using polynomial space can be solved using logspace plus a one-way brute force search of exponentially many solutions (in a fixed ordering, totally independent of the input). The languages in

$PSPACE$ can be decided by a machine such that for all inputs x, the configurations of the valid computation history of the machine appear in lexicographic order, modulo a $O(\log n)$ factor. Furthermore, the result also emphasizes the power of addition, even when we are merely adding one repeatedly.

8.1 Logspace + Polynomial Increment = PSPACE

We have defined what is increment access above. We now define a particular increment access model to be studied.

Definition 8.1 *$L + PI$ is the class of languages accepted by logspace Turing machines with 2-way read-only* **increment access** *to polynomial reference, i.e. the content of the tape is treated as an (initially zero) integer that the logspace machine can increment; that is, add one to the current integer on the tape. We enforce acception/rejection behavior: an increment cannot happen when the integer is all ones; rather, the machine must accept or reject in this position.*

We have strong motivations for defining such a model. First, the ability to read and increment is intuitively more powerful than reading alone, and also less powerful than reading and writing. It is a restrictive form of writing, in which sometimes only one bit is written, and other times all bits change. This model may be construed as

an abstract version of a brute force searching mechanism. Normally when one thinks of brute force search, the classes NP or Δ_2^P come to mind. However, our result shows that if a logspace machine is truly given the capability of accessing all possible solutions to a problem instance, we can exploit our access the solution space to find more than just one solution, or two or three or exponentially many solutions. Rather, we can simulate alternating polynomial time with the solution space and our logspace machine. Recall that alternating polynomial time is equivalent to $PSPACE$ [2].

Theorem 8.1 $PSPACE = L + PI$.

Proof. $L + PI \subseteq PSPACE$ is true since the $L + PI$ model always uses no more than polynomial space.

For the other direction, it suffices to prove that for any $S \in AP$ (alternating polynomial time), we can decide S using a $L + PI$ machine.

For all $S \in AP$, without loss of generality we may consider the logical formula describing S as having polynomially many repeated pairs of existential and universal variables, followed by a polynomial time predicate P. Call these pairs of variables "Σ_2^p-pairs". That is, we can express the decision criteria for S by the following:

For some constants c, d,

$$x \in S \iff (\exists y_1 \forall y_2)(\exists y_3 \forall y_4) \cdots (\exists y_{2|x|^d-1} \forall y_{2|x|^d}) P(x, y_1, y_2, \ldots, y_{2|x|^d}),$$

and each y_i *is such that* $|y_i| \leq |x|^c - 1$.

Observe that the formula described above has $|x|^d$ Σ_2^p-pairs.

First, we describe the design of the reference. Let the running time of predicate P be n^e. The reference will be of length $n^{(c+d)(1+e)}$, storing n^d variables y_i, each of length $n^c - 1$, and an additional string of length $n^{(c+d)e}$. [3] The location for the variable y_k on the reference will start at the index $(k-1)n^c$ and end at index $kn^c - 1$. The remaining (rightmost) bits of the tape will be called *valcomp*, used for the search for a valid computation history of $P(x, y_1, y_2, \ldots, y_{2|x|^d})$.

We now describe some helper functions for the algorithm. Given the reference, the function *zeroes(i)* returns true iff $y_i = y_{i+1} = \cdots = y_n = 0^{n^c}$ and $valcomp = 0^{n^{(c+d)(1+e)}}$. *zeroes* can be implemented using logspace, since the reference is of polynomial length. The command "Increment y_i" increments the reference until some bit in variable block y_i changes, i.e. $y_{i+1}, \ldots, y_n$ and *valcomp* are all zeroes, i.e. *zeroes(i + 1)*.

Let M_P be a polytime machine deciding predicate P in time n^e. The algorithm decides S using possibly exponentially many calls to $M_P(x, y_1, \ldots, y_{2|x|^d})$, for various values of y_i. The logspace machine is used to (1) search for a valid computation history for M_P using x with the current values of y_i, and (2) increment the tape in a way that simulates alternation. Point (2) is achieved using a pointer to the reference

[3] This length is an upper bound on the tape required, but no loss of generality results.

called *current*, which will indicate the current Σ_2^p-pair under consideration, a boolean variable *accept* which represents what the last call to M_P returned (true or false), and a *direction*, which is either left (L) or right (R). The actual procedure implementing point (2) is described below.

Point (1) is done in the following way. Given that $valcomp = 0^{n^{(c+d)(1+e)}}$, we use the logspace machine to determine if, given the current values of y_i (and the fact that we can locate any of them quickly) and x, that the current contents of *valcomp* is a valid computation history of M_P with these particular values as input. If it is, then we set *accept* accordingly. If it is not, we increment *valcomp*, and try again. Since M_P is a polytime machine, we will find an accepting or rejecting valcomp before we have exhausted all possible values of valcomp. However, when a valcomp is found, we increment *valcomp* until it is $1^{n^{(c+d)(1+e)}}$.

The rest of the procedure is given below.

Input: $(\exists y_1 \forall y_2)(\exists y_3 \forall y_4) \cdots (\exists y_{n-1} \forall y_n) F$.

Initially, $current = n$, $direction = R$, and the reference is all zeroes.

Repeat until current < 2:

If $current = n$, then

$accepted$ =1 if $M_P(x, y_1, \ldots, y_{2|x|^d})$ accepts, else it is 0. $direction$ =L.

else if *direction* $= L$ then

If *accepted* then

If $y_{current} \neq 1^{n^c}$ then increment y_2, *direction* $= R$.

else

If $y_{current-1} \neq 1^{n^c}$ then increment y_1, *direction* $= R$.

End if

End if

If *direction* $= R$, then *current = current + 2*, else *current = current - 2*.

End repeat.

Return *accepted*.

Note that the algorithm accepts iff (*accepted* $= 1$) and for all $k = 1, \ldots, |x|^d$, $y_{2k} = 1^{n^c}$. We argue that this condition holds iff

$$(\exists y_1 \forall y_2)(\exists y_3 \forall y_4) \cdots (\exists y_{2|x|^d-1} \forall y_{2|x|^d}) P(x, y_1, y_2, \ldots, y_{2|x|^d}).$$

More precisely, our claim is that when $y_1, \ldots y_{2(k-1)}$ fixed, *accepted = 1, direction* $= L$, and *current* $= 2k$ if and only if $(\exists y_{2k-1} \forall y_{2k}) \cdots (\exists y_{2|x|^d-1} \forall y_{2|x|^d}) P(x, y_1, y_2, \ldots, y_{2|x|^d})$.

If $|x|^d = 1$, (one Σ_2^p-pair) the claim is easy. When y_1 is incremented, $y_2 = 0^{n^c}$. If $y_2 = 1^{n^c}$, then y_2 was incremented 2^{n^c} times, i.e. M_P accepted for all 2^{n^c} values of

y_2, and y_1 never changed during these runs of M_P. Therefore, there exists a y_1 such that for all values of y_2, M_P accepts. Conversely, if such a y_1 exists, then clearly once the algorithm has incremented y_1 to this value, all values for y_2 will accept, so y_2 will reach 1^{n^c} and the algorithm will accept.

Our inductive step is to assume the claim holds for $2k$ variables y_i (i.e. k Σ_2^p-pairs). Suppose at the end of the algorithm,($accepted = 1$) and for all $k = 1, \ldots, n/2$, $y_{2k} = 1^{n^c}$. Let us say the algorithm *almost accepts* at step i of the algorithm when *current* is decremented to 2 in that step, and *accepted* = 1. Our induction hypothesis is that when this happens, given the current values of y_1 and y_2, $(\exists y_3 \forall y_4) \cdots (\exists y_{2|x|^d-1} \forall y_{2|x|^d}) P(x, y_1, y_2, \ldots, y_{2|x|^d})$. Since $y_2 = 1^{n^c}$, we claim that by the properties of the algorithm, it almost accepts for each value of y_2. Furthermore, y_1 is fixed throughout each almost acceptance, and each value of y_2. Hence there exists a y_1 such that for all y_2, almost acceptance occurs.

Conversely, if $(\exists y_1 \forall y_2)(\exists y_3 \forall y_4) \cdots (\exists y_{2|x|^d-1} \forall y_{2|x|^d}) P(x, y_1, y_2, \ldots, y_{2|x|^d})$, almost acceptance will fail to occur until the algorithm increments y_1 to be some value, and then each time y_2 is valued 0^{n^c} to 1^{n^c}, almost acceptance will occur. Thus by induction this direction holds.

□

8.2 Enforcing order on $PSPACE$ computations

We mentioned briefly above that $L + PI$ may be considered as an abstract version of a brute force searching mechanism. It is important to note that the mechanism we have described is somewhat unidirectional over the search space. That is, suppose we replace the polynomial reference with an exponential (in n^c) reference, with one-way read-only access to a blank tape, but our machine can read the reference head's current position on the reference. It is not difficult to show that this model is equivalent to the $L + PI$ model.

But this observation does indicate something very strange about the nature of one-way versus two-way access itself, when our reference size is doubly exponential in our working storage size. When the tape is blank, and can only be accessed it one-way, the only useful information the machine obtains from the reference is the current position of the reference head. However, simply being able to discern this position makes the model as powerful as if the reference was full of information, and could be accessed bidirectionally.

The phenomenon we are observing here (of moving from one-way with a polynomial index to two-way with exponential information) is due to the weakness of the logspace machine. Even with a full exponential length plate of data at its side, it will cycle after reading only a polynomial amount of it in one direction. Therefore, a

logspace machine in this situation must rely on the reference to indicate its position on the tape (it may verify over time that the reference is not lying about the position, but it cannot possibly store this position itself). Our result shows that *all* the logspace machine needs to know in order to simulate $PSPACE$ is its current position on the reference, and any other information the tape gives it is extraneous (it can be verified by the logspace machine itself).

In other words, we can give a weaker $PSPACE$-complete problem about finite automata, which is an improvement of Hunt's result [8] that the emptiness problem for two-way DFAs is $PSPACE$-complete:

Theorem 8.2

$$2DFABIN = \{\langle M\rangle : (M \text{ is a } 2DFA) \wedge (0^{|M|}\#0^{|M|/2-1}1\#0^{|M|/2-2}10\#\cdots\#1^{|M|/2} \in L(M))\}$$

is $PSPACE$-complete.

Proof. This problem is $PSPACE$ hard because we can reduce from any $L + PI$ computation to it, using the same techniques as given in the previous section on automata. Given a $L + PI$ machine G and input x, without loss of generality it can be assumed that when $G(x)$ increments its reference, it moves the reference head to the leftmost tape square, without recording anything new to worktape, and without moving its input head in the process. (This follows from the proof that $L + PI =$

$PSPACE$.) We reduce the logspace portion of $G(x)$ to a two-way automaton M, such that $G(x)$ accepts if and only if there is a string that M accepts. We construct M such that:

In configuration C, $G(x)$ increments its tape and moves its tape head to the leftmost square $\iff$ M in its own state C moves to the right until it reaches a #, then it moves right once more; otherwise, the simulation of $G(x)$ using M is straightforward.

It is in $PSPACE$: initially, a worktape will consist of the bit string $0^{|M|}$, plus a #. The $2DFA$ M is simulated on that string, until M wishes to advance its head to the right beyond the #. Then, the string on the tape is incremented, the worktape head moves to the leftmost tape square, and the simulation continues. When M wishes to advance to the left beyond the current string on tape, the string on the tape is decremented, the worktape head moves to the rightmost tape square, and the simulation continues. As a matter of course, the end boundaries of the worktape are also enforced: if the string on worktape is $0^{|M|}$, requests by M to move left beyond this string are denied; if the string is $1^{|M|}$, requests to move right beyond this string are denied.

□

Given a $2DFA$ M, if there exists an NP procedure which proves that M accepts a

very natural and highly redundant exponential length string, then $NP = PSPACE$. Compare this with the earlier result that the non-emptiness problem for strings of length $\leq |M|$ is NP-complete. In one case, we are asked to find any string at all of length $|M|$. In the other, we are asked to determine acceptance for a particular string of length $2^{|M|}$. Is it more difficult to find a short proof, or to verify a very long one? At this stage of our work, we cannot say.

8.3 A deterministic formalism for P^{NP}

We have seen that in terms of complexity, logspace is actually a tremendous amount of working storage to give to a machine that is performing brute-force search of a polynomial-sized solution space. However, reducing the working storage to less than logspace may be too restrictive for our goal of finding models characterizing classes between NP and $PSPACE$.

Rather than reducing the amount of working storage, we should limit the length of time for which bits can be held in storage. In our proof that $L + PI = PSPACE$, it was crucial that we could save the pointers for much longer than polynomial time, waiting until some part of the reference consists of all ones.

This motivates us to restrict the time for which pointers can stay in working storage, say polynomial time. We would cripple our increment model if it were unable to

keep its working storage, state, and head positions after it incremented the reference– if the logspace machine resets to its initial configuration every time an increment is performed. Our model would lend itself to parallelization, since with such a restriction one can execute the machine with its reference set to y, somewhat independently of a run where the reference is set to some $z \neq y$.

We informally define $L + PIR$ as the class of $L + PI$ machines with the obvious restriction: when the logspace machine enters an increment state, it writes blanks on every square of its worktape, moves its tape heads to the leftmost positions, and resets to the initial state. The main result of this section is that given this "reset" restriction, the model is indeed crippled, but its complexity falls to an unexpected level of the polynomial time hierarchy.

Theorem 8.3 $L + PIR \subseteq P^{NP}$.

Proof. Let $M_{\log}(\cdot, \cdot)$ denote the **logspace portion** of an arbitrary $L + PIR$ machine M. That is, $M_{\log}$ is a logspace machine with read-only inputs x and a polynomially related reference y, with final states being accepting, rejecting, or incrementing. Furthermore, $(\forall x, y \in \Sigma^*)[M(x)$ accepts/rejects/increments on reference $y \iff M_{\log}(x, y)$!accepts/rejects/increments. So the logspace portion represents what M does on a fixed reference.

Note that specifying the logspace portion of the machine is sufficient for specifying an M. To show the theorem, it suffices to demonstrate that a particular language $\mathcal{L}$ is both in P^{NP}, and $L + PIR$-complete under logspace reductions. Define

$$ISAT := \{\langle M^k_{\log}, x, 1^{n^k}\rangle : (\exists y : |y| \leq |x|^k)(\forall z < y)[M(x, z) \text{ increments} \wedge M(x, y) \text{ accepts}\},$$

where $M^k_{\log}$ are logspace machines executing in n^k time.

Lemma 8.1 *ISAT is $L + PIR$-complete under logspace reductions.*

Proof. Let $M_{\log}$ be the logspace part of a $L + PIR$ machine M, and x be an input. By the criterion given for acceptance and rejection, we know there is at least one string y that, when it is set as the reference, $M_{\log}(x, y)$ either accepts or rejects. But for a deterministic execution of M on x, this reference y may never be reached; that is, it may accept or reject on an $z < y$.

To be precise, $M(x)$ accepts (rejects) if and only if there is a reference y such that for all $z < y$, an increment is called at the end of each z computation, and on y the machine accepts, i.e.

$$M(x) \text{ accepts} \iff (\exists y : |y| \leq |x|^k)(\forall z < y)[M_{\log}(x, y) \text{ accepts} \wedge M_{\log}(x, y) \text{ increments}]$$

Using a standard padding technique, $ISAT$ as defined above is $L + PIR$-complete under logspace reductions.

□

Lemma 8.2 $ISAT \in P^{NP}$.

Proof. On an input $\langle M^k_{\log}, x, 1^{n^k}\rangle$, the following P^{NP} procedure will test at most n^k values for y with the predicate $R(x,y) = (\forall z \leq y)[M^k_{\log}(x,z) = \text{increment}]$, which can be tested in one step using an NP oracle.

Given a string $x = \sigma_1 \cdots \sigma_m \in \Sigma^*$, for $i = 1, \ldots, m$ we define $x[i] := \sigma_1 \cdots \sigma_i$.

Algorithm($\langle M^k_{\log}, x, 1^{n^k}\rangle$):

$u := \epsilon$, $v := 10^{n^k - 1}$.

While $v \neq \epsilon$,

Set $y := uv$.

If $R(x,y)$ then

If $M^k_{\log}(x,y)$ accepts/rejects then **accept**/**reject**, else $u := u1$.

else $u := u0$.

End if

$v := v[|v| - 1]$

End while

It is clear that the while loop is finished in n^k iterations. Besides the oracle, the algorithm executes in deterministic polynomial time, using only a polynomial number of queries.

Suppose $M(x)$ accepts/rejects. Then there exists a unique y with $M^k_{\log}(x, y)$ accepting/rejecting, with all $z < y$ having $M(x, z)$ increment. Observe that if $R(x, y')$ is true, then y is greater than or equal to y'. If $R(x, y')$ is false, then $y' \leq y$. This allows us to perform a binary search technique on the exponential number of possible y values. If $R(x, y)$ is true in the ith iteration ($i = 1, \ldots, n^k$), y is increased by $2^{n^k - i}$. If $R(x, y)$ is false, y is decreased by $2^{n^k - i}$. By repeatedly querying a "midpoint" of the current solution space (such as $10^{n^k - 1}$ in the list of strings from 0^{n^k} to 1^{n^k}), we half the number of solutions under consideration during each iteration of the while loop.

Conversely, if the simulation accepts/rejects, then the final y value obtained is such that $M^k_{\log}(x, y)$ accepts/rejects, and the $z < y$ all increment, by definition of $R(x, y)$.

□

Theorem 8.4 *$P^{NP} \subseteq L + PIR$.*

Proof. Our goal is to build a brute-force simulation for P^{NP} machines. A natural way to simulate machine models involves verifying valid computation histories that

claim the model accepts or rejects on the given input. It will be helpful then to find a way to characterize valid computation histories for P^{NP}.

We start by extending the notion of a valid computation history for nondeterministic machines, to include tokens which indicate rejection.

Let N be execute in n^k nondeterministic time, and assume without loss of generality that $1^{n^{2k}}$ is not a valid computation of N for any x.

Definition 8.2 *The set of* **extended valcomps of** N **on** x *is* $\texttt{VALCOMP}^*(N,x) := \texttt{VALCOMP}(N,x) \cup \{1^{|x|^{2k}}\}$*, where* $\texttt{VALCOMP}(N,x)$ *is the well-known set of valid computation histories of* N *on* x*.* [4]

One may argue that $1^{|x|^{2k}}$ is not a well-defined token. Indeed, it could very well represent the rejection of any string of length $|x|$. Those who would speculate in this manner should be patient.

This definition is generalized to P^{NP} machines. Let x be an input, and M^N be the P^{NP} machine under consideration, where the query machine $M^?$ and N execute in

[4]For the sake of brevity we do not formally define what a valid computation history is, nor do we define the set of them. For more examples of applications using VALCOMPS, see Kozen's book, or Hopcroft and Ullman [9], [7].

n^k and n^c time, respectively. We assume $M^?$ has a separate query tape with content y, and entering state $q^?$ of $M^?$ represents querying if $N(y)$ accepts or not.

Suppose the configurations of $M^?(x)$ are (ordered from initial to final) $C_1, C_2, \ldots, C_{n^k}$. Set the configurations of $M^?(x)$ that are in state $q^?$ as $C_{i_1}, \ldots, C_{i_Q}$, and the queries as $y_1, \ldots, y_Q$. We label them in the order they appear; that is, C_{i_j} queries N about y_j, and $C_{i_j} \vdash^* C_{i_{j+1}}$. Assume C_{n^k} is a final configuration.

Definition. We define the extended valcomp of $M^N(x)$ as

$\texttt{VALCOMP}^*(M^N, x) :=$

$$C_1 \cdots C_{i_1} \texttt{VALCOMP}^*(N, y_1) C_{i_1+1} \cdots C_{i_2} \texttt{VALCOMP}^*(N, y_2) C_{i_2+1} \cdots$$
$$\cdots C_{i_Q} \texttt{VALCOMP}^*(N, y_Q) C_{i_Q+1} \cdots C_{n^k}$$

Intuitively, we thread the VALCOMPs of the machine N within the string of configurations of machine $M^?$. Note that $\texttt{VALCOMP}^*(M^N, x)$ may be a set of several strings, since $\texttt{VALCOMP}^*(N, y_i)$ may obviously be a set.

As it stands, there is no reason to assume that these extended valid computations are of any validity in themselves. Their certification arises when they are analyzed with a brute-force search mechanism that resets itself after each solution check.

$SIM_{M,N}(x)$:

Try to verify that the reference is in $\texttt{VALCOMP}^*(M^N, x)$ via this process:

verify transitions for M the usual way. Verify transitions for a query $N(y_i)$ in the usual way as well, except for all $N(y_i)$, interpret $1^{|y_i|^{2c}}$ as a valcomp that represents a "no" answer to the query. When a mistake is found, increment the reference and reset. Return the outcome (accept or reject) of the first string found to be in $\texttt{VALCOMP}^*(M^N, x)$.

It is easy to see that:

(1) This procedure can be performed by a logspace machine with increment access to a polynomial length reference.

(2) $SIM_{M,N}(x)$ returns an answer if and only if its current reference is a member of $\texttt{VALCOMP}^*(M^N, x)$.

(3) If $M^N(x)$ accepts (rejects), then any element of $\texttt{VALCOMP}^*(M^N, x)$ will be considered an accepting (rejecting) computation by $SIM_{M,N}(x)$.

Of course, these facts disregard the actual validity of strings in $\texttt{VALCOMP}^*(M^N, x)$. They merely show that $SIM_{M,N}(x)$ returns when it finds a member of a restricted subset of references, though they may represent invalid computations (that is, $1^{|y_i|^{2k}}$ may appear in the place of an accepting valcomp of $N(y_i)$).

Now we prove: if $SIM_{M,N}(x)$ accepts (rejects) when the reference is a member of $\texttt{VALCOMP}^*(M^N, x)$, then $M^N(x)$ accepts (rejects). First, observe that for a given

x, there is a unique list of queries $y_1, \ldots, y_Q$ made by $M^?(x)$. This is because M is deterministic, and the query y_i has one possible outcome.

Let us recall the expression

$C_1 \cdots C_{i_1} \texttt{VALCOMP}^*(M, y_1) C_{i_1+1} \cdots\cdots\cdots C_{i_Q} \texttt{VALCOMP}^*(N, y_2) C_{i_Q+1} \cdots C_{n^k}$ as $\texttt{VALCOMP}^*(M^N, x)$.

First, $SIM_{M,N}(x)$ increments the reference until it is of the form $C_1 C_2 \cdots C_{i_1} 0000 \cdots 0000$. Then the incremental search over all possible strings of length n^{2c} for a member of $\texttt{VALCOMP}^*(N, y_1)$ continues until either an accepting valcomp is found (and the query answer is "yes"), or $1^{|y_1|^{2c}}$ appears (and the query answer must thus be "no", as all other possible members will have been already considered). The verification procedure then returns to M, incrementing until $C_{i_1+1} \cdots C_{i_2}$ appear, and the verification of $N(y_2)$ begins. The same desired properties (that the rest of the reference other than what $SIM_{M,N}$ has verified is $0 \cdots 0$ when a query is made, that the "no" inference for a query is valid) hold for all other successive configurations under consideration.

Without loss of generality, every configuration of $M^N(x)$ always has a successor (or is final). Therefore, $SIM_{M,N}$ never increments in a way that damages our valcomp construction's progress: there is always a valid configuration following the configurations it has constructed so far, which is found by incrementing before an overflow can occur.

□

Several remarks are in order. Our above result equates a complexity class defined in terms of an *abstract, non-physical* model, with one defined by *concrete, mechanical* model. The abstract model executes in polynomial time but receives advice about difficult problems from an oracle. The concrete model uses exponential time but a highly restrictive brute-force search. We have classified what appears (on the face of it) as an extremely artificial and contrived model of computation, using a model that follows a completely deterministic and physical procedure. For those concerned about the real-world relevance of particular portions of complexity theory, this should be a pleasant surprise.

Next, consider a three-valued predicate R that can be evaluated in logspace. The previous theorem implies a very natural characterization of a well-studied complexity class between *co-*NP (NP) and Σ_2^p (Π_2^p).

Corollary 8.1 $\{x : (\exists^p y)(\forall z < y)[R(x,y) = 0 \wedge R(x,z) = 1]\}$ *and*

$\{x : (\forall^p y)(\exists z < y)[R(x,y) = 0 \wedge R(x,z) = 1]\}$ *are complete for* P^{NP}.

Compare and contrast the following three results:

$\{x : (\forall^p z)[R(x,z) = 1]\}$ is *co-*NP complete.

$\{x : (\exists^p y)(\forall z < y)[R(x,y) = 0 \wedge R(x,z) = 1]\}$ is P^{NP} complete.

$\{x : (\exists^p y)(\forall^p z)[R(x,y,z) = 1]\}$ is Σ_2^p complete.

The complete problem for P^{NP} has the two quantifiers as Σ_2^p, placing an ordering restriction on the universal quantifier. However, it retains the binary predicate as found with the *co-NP* complete problem.

We remark that a similar analogy can be drawn between NP, P^{NP}, and Π_2^p. We conjecture that this result may be generalized in a straightforward way to Σ_k^p, Δ_k^p, and Π_{k+1}^p.

9 Conclusion

A new formulation of the basic complexity classes has been presented. It is significant for several reasons. It builds a new picture of the relative power of each class, and further empirical evidence for why we should believe each class is different from the others in question. Every logspace machines is equivalent to some k-head-DFA, where the heads may know their positions on the input. What is the complexity of a k-DFA with sequential read-only access to a short arbitrary reference, versus random read-only access? This is the essence of the NL ?= NP problem. What is the resulting complexity when a k-DFA has append access to a blank short reference, versus random read-only access to any fixed short reference? Herein lies the P vs. NP problem. When there are no access restrictions to the reference, the model computes $PSPACE$. Our model translates these fundamental questions into formal questions in system security and protection: for example, if $P = NP$, then a user with logspace can (given append access to massive storage) reconstruct some massive reference to which it does not have random read-only access. The increment models are interesting in their own right, demonstrating that access to a exponential time counter is enough to capture polynomial space in general, and resetting the machine on a counter change gives the oracle-based complexity class P^{NP}.

On a philosophical level, access complexity is a fundamentally different way to

think of well-studied objects. One may think of computation universally as a mechanical (and in the abstract, sometimes physically unrealizable) search for an answer to a well-defined problem. The theory proposed in this work advocates a view of complexity which distills the *robustness* of the most studied classes from the *search mechanism*, the component that truly indicates the differences between the classes. That is, the arbitrary logspace Turing machine gives our models the power of reductions (and thus complete sets), and the various access permissions determine how solutions are found. One might say that while the robustness of the classes makes them difficult to separate, the wildly varying range of searches they employ continue to convince us that they must be separable. As we have seen, for NP this robustness can be whittled down to an logtime machine with three alternations. Continuing work is being done on finding weaker machine models to substitute for logspace in the given formulations, with the motivation being that if we shrink the contribution given by robustness, the search mechanisms may be more pronounced.

This leads to our final point: that the pronounced dichotomy between robustness and access may have further implications on the formal level, in the way of proof methods. Separating the classes under these new models may or may not be an easier task; however, studying the power of access permissions appears to admit different proof methods than those found in traditional complexity theory. With the alleged

disparities of the classes pointed out by markedly more intuitive notions (reading, writing, increment, sequential and random access) it seems clear that new results concerning disparities between classes (conditionally or unconditionally) should arise from further study of the ideas presented.

10 References

[1] D. Barrington, N. Immerman, H. Straubing, "On Uniformity within NC^1", *Journal of Computer and System Sciences*, 41:3, 274-306, 1990.

[2] A. Chandra, D. Kozen, L. Stockmeyer, "Alternation," *Journal of the ACM*, 28:114-133, 1981.

[3] S. Cook, "Short propositional formulas represent nondeterministic computations," *Information Processing Letters*, 26:269-270, 1988.

[4] J. Hartmanis, "On nondeterminacy in simple computing devices," *Acta Informatica*, 1, 336-344, 1972.

[5] J. Hartmanis and R. Stearns, "On the computational complexity of algorithms," *Transactions of the AMS*, 117: 285-306, 1965.

[6] J. Hartmanis, P. Lewis, R. Stearns, "Hierarchies of memory-limited computations," *Proc. 6th Annual IEEE Symp. on Switching Circuit Theory and Logic Design*, 179-

190, 1965.

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

[8] H. Hunt, *On the Time and Tape Complexity of Languages*, Doctoral Thesis, Dept. of Computer Science, Cornell University, Ithaca, NY, 1972.

[9] D. Kozen, *Automata and Computability*, Springer-Verlag, New York, NY, 1994.

[10] D. Kozen, "Lower bounds for natural proof systems," *Proc. 18th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, 254-266.

[11] N. Immerman, "Nondeterministic space is closed under complement," *SIAM Journal of Computing*, 17: 935-938, 1988.

[12] N. Immerman and S. Landau, "The Complexity of Iterated Multiplication," *Information and Computation*, 116:103-116, 1995.

[13] M. Rabin and D. Scott, "Finite automata and their decision problems," *IBM Journal of Research* 3:115-125, 1959.

[14] S. Rudich, et. al. *Computational Complexity Lecture Notes*, Carnegie Mellon University, 2000.

[15] W. Savitch, "Relationship between nondeterministic and deterministic tape classes," *Journal of Computer and System Sciences*, 4: 177-192, 1970.

[16] M. Sipser, "Halting space-bounded computations," *Theoretical Computer Science*, 10: 335-338, 1980.

[17] R. Szelepcseyni, "The method of forcing in automata," *Bulletin of the EATCS*, 33: 96-100, 1987.

[18] B. Trakhtenbrot, "A survey of Russian approaches to *perebor* (brute-force search) algorithms", *Annals of the History of Computing*, 6: 384-400, 1984.