YALE UNIVERSITY
Department of Computer Science

EXPRESSIBILITY AND PARALLEL COMPLEXITY

Neil Immerman

YALEU/DCS/TR-546
June 1987
Revised August 1988

Expressibility and Parallel Complexity*

Neil Immerman†

Computer Science Department
Yale University
New Haven, CT 06520

Abstract

We show that the time needed by a concurrent read, concurrent write, parallel random access machine (CRAM) to check if an input has a certain property is the same as the minimal depth of a first-order inductive definition of the property. This in turn is equal to the number of 'iterations' of a first-order sentence needed to express the property.

The second contribution of this paper is the introduction of a purely syntactic uniformity notion for circuits. We show that an equivalent definition for the uniform circuit classes AC^i, $i \geq 1$ is given by first-order sentences 'iterated' $\log^i n$ times. Similarly, we define uniform AC^0 to be the first-order expressible properties (which in turn is equal to constant time on a CRAM by our main theorem).

A corollary of our main result is a new characterization of the Polynomial-Time Hierarchy (PH): PH is equal to the set of languages accepted by a CRAM using exponentially many processors and constant time.

1 Introduction

Parallel time on a random access machine has a surprisingly simple mathematical definition involving well studied objects of mathematical logic. We

*This paper will appear in *SIAM Journal of Computing*.

†Research supported by the Mathematical Sciences Research Institute, Berkeley, CA, and by NSF Grants DCR-8603346 and CCR-8806308.

show that the time needed by a concurrent read, concurrent write, parallel random access machine (CRAM) to check if an input has a certain property is the same as the minimal depth of a first-order inductive definition of the property. This in turn is equal to the number of 'iterations' of a first-order sentence needed to express the property.

We now state our main result. (See Section 2 for relevant definitions. In particular, the iteration of a first-order sentence is defined in Subsection 2.2, and the CRAM is defined in Subsection 2.3. The definition of the CRAM differs from the standard definition of the CRCW PRAM in [23] only in that a processor may shift a word of local memory by any polynomial number of bits in unit time. It follows from our results that for parallel time greater than or equal to $\log n$ there is no distinction between the models with and without the Shift instruction.)

Theorem 1.1 *Let S be a set of structures of some vocabulary τ, for example: S is a set of boolean strings, or a set of graphs, etc. For all polynomially bounded, parallel time constructible $t(n)$, the following are equivalent:*

1. *S is recognizable by a CRAM in parallel time $t(n)$, using polynomially many processors.*
2. *There exists a first-order sentence φ such that the propery S for structures of size at most n is expressed by φ iterated $t(n)$ times.*
3. *S is definable as a uniform first-order induction whose depth, for structures of size n, is at most $t(n)$.*

For $t(n) \geq \log n$, the equivalence of (1) and (2) in Theorem 1.1 may also be obtained by combining a result of Ruzzo and Tompa relating CRAMs to alternating Turing machines [23, Theorem 3], together with a result of ours relating alternating Turing machines to first-order expressibility [15, Theorem B.4]. In order to prove the theorem for $t(n) < \log n$ we were forced to modify the models slightly, adding the Shift operation to the CRAMs and adding BIT as a new logical relation to our first-order language (see Section 2). We believe that the naturalness of Theorem 1.1 justifies these modifications.

This paper is organized as follows. In Section 2 we give all relevant definitions. In Section 3 we prove our main result. In Section 4 we give a more detailed analysis of the bounds in Theorem 1.1. We show that the number of distinct variables in a first-order inductive definition is closely tied to the number of processors in the corresponding CRAM.

Until now, a principal unaesthetic feature of the theory of complexity via boolean circuits was that one had resorted to Turing machines to define the uniformity conditions for circuits [21]. As a corollary to Theorem 1.1, we obtain a purely syntactic uniformity notion for circuits. In Section 5 we describe this result as well as other relations between circuits and first-order complexity.

As another corollary to Theorem 1.1, we present in Section 6 a new characterization of the Polynomial-Time Hierarchy (PH): PH is equal to the set of languages recognized by a CRAM using exponentially many processors and constant time. In Section 7 we give some suggestions for future work in this area.

2 Background and Definitions

2.1 First-Order Logic

We begin this section by making some precise definitions concerning first-order logic. For more information see [6].

A *vocabulary* $\tau = \langle \underline{R}_1^{a_1} ... \underline{R}_k^{a_k}, \underline{c}_1 ... \underline{c}_r \rangle$ is a tuple of relation symbols and constant symbols. $\underline{R}_i^{a_i}$ is a relation symbol of arity a_i. In the sequel we will usually omit the superscripts and the underlines to improve readability. A finite *structure* of vocabulary τ is a tuple, $\mathcal{A} = \langle \{0, 1, \ldots, n-1\}, R_1^{\mathcal{A}} ... R_k^{\mathcal{A}}, c_1^{\mathcal{A}} ... c_r^{\mathcal{A}} \rangle$, consisting of a universe $|\mathcal{A}| = n = \{0, \ldots, n-1\}$ and relations $R_1^{\mathcal{A}} ... R_k^{\mathcal{A}}$ of arities $a_1, \ldots, a_k$ on $|\mathcal{A}|$ corresponding to the relation symbols $\underline{R}_1^{a_1} ... \underline{R}_k^{a_k}$ of τ, and constants $c_1^{\mathcal{A}} ... c_r^{\mathcal{A}}$ from $|\mathcal{A}|$ corresponding to the constant symbols $\underline{c}_1 ... \underline{c}_r$ from τ.

For example, a graph on n vertices, $G = \langle \{0...n-1\}, E \rangle$, is a structure whose vocabulary $\tau_0 = \langle \underline{E}^2 \rangle$ has a single binary relation symbol. Similarly, a binary string of length n is a structure $S = \langle \{0...n-1\}, M \rangle$ whose vocabulary $\tau_1 = \langle \underline{M}^1 \rangle$ consists of a single unary relation symbol. Here the i^{th} bit of S is 1 iff $S \models M(i)$.

Let the symbol '$\leq$' denote the usual ordering on the natural numbers. We will include $\leq$ as a logical relation in our first-order languages. This seems necessary in order to simulate machines whose inputs are structures given in some order. It is convenient to include logical constant symbols, $0, 1, \ldots$, refering to the zeroth, first, etc. elements of the universe, respectively. (If the universe is smaller than a given constant then interpret that constant as 0.) We also include the logical predicate BIT, where $\text{BIT}(x, y)$ holds iff

the x^{th} bit in the binary expansion of y is a one.[1]

We now define the *first-order language* $\mathcal{L}(\tau)$ to be the set of formulas built up from the relation and constant symbols of τ and the logical relation and constant symbols, $=, \leq, \text{BIT}, 0, 1, \ldots$, using logical connectives, $\wedge, \vee, \neg$, variables, $x, y, z, \ldots$, and quantifiers, $\forall, \exists$.

We will think of a *problem* as a set of structures of some vocabulary τ. It suffices to only consider problems on binary strings, but it is more interesting to be able to talk about other vocabularies, e.g. graph problems, as well. For definiteness, we will fix a scheme for coding an input structure as a binary string. If $\mathcal{A} = \langle \{0, 1, \ldots, n-1\}, R_1^{\mathcal{A}} \ldots R_k^{\mathcal{A}}, c_1^{\mathcal{A}} \ldots c_r^{\mathcal{A}} \rangle$, is a structure of type τ, then $\mathcal{A}$ will be encoded as a binary string $\text{bin}(\mathcal{A})$ of length $I(n) = n^{a_1} + \cdots + n^{a_k} + r\lceil \log n \rceil$, consisting of one bit for each a_i-tuple, potentially in the relation R_i, and $\lceil \log n \rceil$ bits to name each constant, c_j. Thus we reserve n to indicate the size of the universe of the input structure. $I(n)$, the length of $\text{bin}(\mathcal{A})$, is polynomially related to n, and in the case where τ consists of a single unary relation – i.e. inputs are binary strings – $I(n) = n$.

Define the complexity class FO to be the set of all first-order expressible problems. We will see in Section 5 that FO is a uniform version of the circuit class AC^0. (See also [2] where it is shown that FO is equal to deterministic log time uniform AC^0.)

Example 2.1 *An example of a first-order expressible property is addition.*[2] *In order to turn addition into a yes/no question we can let our input have the vocabulary $\tau_a = \langle A, B, k \rangle$ consisting of two unary relations and a constant symbol. In a structure $\mathcal{A}$ of vocabulary τ_a, the relations A and B are binary strings of length $n = |\mathcal{A}|$. We'll say that $\mathcal{A}$ satisfies the addition property if the k^{th} bit of the sum of A and B is one.*

In order to express addition we will first express the carry bit,

$$CARRY(x) \equiv (\exists y < x)[A(y) \wedge B(y) \wedge (\forall z.y < z < x)A(z) \vee B(z)]$$

Then with $\oplus$ standing for exclusive or, we can express PLUS,

$$PLUS(x) \equiv A(x) \oplus B(x) \oplus CARRY(x)$$

Thus the sentence expressing the addition property is PLUS(k).

[1] The relation BIT is crucial for the truth of Theorem 1.1, when $t(n) < \log n$, and for the plausibility of Definition 5.5.

[2] This is a standard construction, see e.g. [23].

2.2 Iterating First-Order Sentences

To describe properties that are not in AC^0, we need languages that are more expressive than FO. We now recall the definition of the complexity classes FO[$t(n)$][3]. Intuitively, FO[$t(n)$] consists of those problems that may be described by a first-order sentence 'iterated $t(n)$ times.'

Let x be a variable and M a quantifier free formula. We will use the notation $(\forall x.M)\psi$ - read, "for all x such that M, ψ," - to abbreviate $(\forall x)(M \rightarrow \psi)$. Similarly we will write $(\exists x.M)\psi$ - read, "there exists an x such that M, ψ," - to abbreviate $(\exists x)(M \wedge \psi)$. We will call the expressions $(\forall x.M)$ and $(\exists x.M)$ *restricted quantifiers.* Let a *quantifier block* be a finite sequence of restricted quantifers: $\mathrm{QB} = (Q_1 x_1.M_1)\ldots(Q_k x_k.M_k)$. We will use the notation $[\mathrm{QB}]^t$ to denote the quantifier block QB repeated t times. I mean this literally:

$$[\mathrm{QB}]^t = \underbrace{\mathrm{QB}\,\mathrm{QB}\,\mathrm{QB}\cdots\mathrm{QB}}_{t \text{ times}}$$

Note that for any quantifier free formulas $M_0, M_1, \ldots, M_k \in \mathcal{L}(\tau)$, and any $i \in \mathbf{N}$, the expression $[\mathrm{QB}]^i M_0$ is a well formed formula in $\mathcal{L}(\tau)$.

Definition 2.2 *Let $t : \mathbf{N} \rightarrow \mathbf{N}$ be any function, and let τ be any vocabulary. A set C of structures of vocabulary τ is a member of FO[$t(n)$] iff there exists a quantifier block QB and a quantifier free formula M_0 from $\mathcal{L}(\tau)$, such that if we let $\varphi_n = [QB]^{t(n)} M_0$, for $n = 1, 2, \ldots$, then for all structures G of vocabulary τ with $|G| = n$,*

$$G \in C \Leftrightarrow G \models \varphi_n .$$

A more traditional way to iterate formulas is by making inductive definitions, [20,16]. Let IND-DEPTH[$t(n)$] be the set of problems expressible as a uniform induction that requires depth of recursion at most $t(n)$ for structures of size n. In [11], Harel and Kozen introduce a programming language called IND which is closely tied to inductive definitions. They prove that the execution time for their IND programs is equal to the depth of the inductive definitions that describe the programs' input output behavior. Let IND-TIME[$t(n)$] be the set of languages accepted by IND programs using $O[t(n)]$ steps for inputs of size n. Then:

[3]The notation FO[$t(n)$] was first used in [18]; however, the same classes were defined in [16] using the notation IQ[$t(n)$], standing for "iterated queries." See [19] for a survey of descriptive complexity.

Fact 2.3 *[11] For all $t(n)$,*

$$\text{IND-TIME}[t(n)] = \text{IND-DEPTH}[t(n)] .$$

This fact together with Theorem 1.1 shows that there is a simple, high level programming language for which time corresponds exactly to time on a CRAM. In the remainder of this paper we write $\text{IND}[t(n)]$ to signify $\text{IND-TIME}[t(n)]$ as well as $\text{IND-DEPTH}[t(n)]$.

The following fact relates $\text{IND}[t(n)]$ to $FO[t(n)]$. This fact follows easily from Moschovakis' Canonical Form for Positive Formulas, [20].

Fact 2.4 *[16, Corollary 5.8], cf. [20] For all $t(n)$,*

$$IND[t(n)] \subseteq FO[t(n)] .$$

(In particular, a property in $IND[t(n)]$ is expressible as a $FO[t(n)]$ property in which $M_0 \equiv$ false, cf. Definition 2.2.)

Example 2.5 *We show how to transfer a $\log n$ depth inductive definition of the transitive closure of a graph to an equivalent $FO[\log n]$ definition.*

Let E be the edge predicate for a graph G with n vertices. We can inductively define E^*, the reflexive,transitive closure of G, as follows:

$$E^*(x,y) \equiv x = y \vee E(x,y) \vee (\exists z)(E^*(x,z) \wedge E^*(z,y)) .$$

Let $P_n(x,y)$ mean that there is a path of length at most n from x to y. Then we can rewrite the above definition of E^* as:

$$P_n(x,y) \equiv x = y \vee E(x,y) \vee (\exists z)(P_{n/2}(x,z) \wedge P_{n/2}(z,y)) .$$

This can be rewritten:

$$P_n(x,y) \equiv (\forall z.M_1)(\exists z)(P_{n/2}(x,z) \wedge P_{n/2}(z,y))$$

where $M_1 \equiv \neg(x = y \vee E(x,y))$. Note that there is no free occurrence of the variable z after the $\forall z$ quantifier. Thus in this case $(\forall z.M_1)\alpha$ is equivalent to $(M_1 \rightarrow \alpha)$. Next,

$$P_n(x,y) \equiv (\forall z.M_1)(\exists z)(\forall uv.M_2)(P_{n/2}(u,v))$$

where $M_2 \equiv (u = x \wedge v = z) \vee (u = z \wedge v = y)$. Now,

$$P_n(x,y) \equiv (\forall z.M_1)(\exists z)(\forall uv.M_2)(\forall xy.M_3)(P_{n/2}(x,y))$$

where $M_3 \equiv (x = u \wedge y = v)$. *Thus,*

$$P_n(x,y) \equiv [QB]^{\lceil \log n \rceil}(P_1(x,y))$$

where $QB = (\forall z.M_1)(\exists z)(\forall uv.M_2)(\forall xy.M_3)$. *Note that*

$$P_1(x,y) \equiv [QB](false)$$

It follows that

$$P_n(x,y) \equiv [QB]^{\lceil 1+\log n \rceil}(false)$$

and thus $E^* \in FO[\log n]$ *as claimed.*

2.3 Concurrent Random Access Machines

We define the concurrent random access machine (CRAM) to be essentially the concurrent read, concurrent write parallel random access machine (CRCW PRAM) described in [23]. A CRAM is a synchronous parallel machine such that any number of processors may read or write into any word of global memory at any step. If several processors try to write into the same word at the same time, then the lowest numbered processor succeeds.[4] In addition to assignments, the CRAM instruction set includes addition, subtraction, and branch on less than. Each processor also has a local register containing its processor number.

The difference between the CRAM and the CRCW PRAM described in [23] is that we also include a Shift instruction. Shift(x,y) causes the word x to be shifted y bits to the right. Without Shift, CRAM$[t(n)]$ would be too weak to simulate $FO[t(n)]$ for $t(n) < \log n$. The reason behind the Shift operation for CRAMs and the corresponding BIT predicate for first-order logic is that each bit of global memory should be available to every processor in constant time.

Let CRAM$[t(n)]$ be the set of problems accepted by a CRAM using polynomially many processors and time $O[t(n)]$. Recall that we encode an input structure $\mathcal{A} = \langle \{0,1,\ldots,n-1\}, R_1^{\mathcal{A}} \ldots R_k^{\mathcal{A}}, c_1^{\mathcal{A}} \ldots c_r^{\mathcal{A}} \rangle$, as the binary string $\mathrm{bin}(\mathcal{A})$ of length $I(n) = n^{a_1} + \cdots + n^{a_k} + r\lceil \log n \rceil$, Where a_i is the arity of the i^{th} input relation. The input string is placed one bit at a time in the first $I(n)$ global memory locations.[5]

[4] This is the 'priority write' model. Our results remain true if instead we use the 'common write' model, in which the program guarantees that different values will never be written to the same location at the same time. See Corollary 3.4.

[5] We show in Corollary 3.4 that if placement of the input is varied, e.g. if the first

3 Proof of the Main Theorem

Theorem 1.1 follows immediately from three containments: Fact 2.4, and the following two lemmas:

Lemma 3.1 *For any polynomially bounded $t(n)$ we have,*

$$CRAM[t(n)] \subseteq IND[t(n)]$$

Proof We want to simulate the computation of a CRAM M. On input $\mathcal{A}$, a structure of size n, M runs in $t(n)$ synchronous steps, using $p(n)$ processors, for some polynomial $p(n)$. Since the number of processors, the time, and the memory word size are all polynomially bounded, we need only a constant number of variables $x_1, \ldots, x_k$, each ranging over the n element universe of $\mathcal{A}$, to name any bit in any register belonging to any processor at any step of the computation. We can thus define the contents of all the relevant registers for any processor of M, by induction on the time step.

We now specify the CRAM model more precisely. We may assume that each processor has a finite set of registers including the following, Processor: containing the number between 1 and $p(n)$ of the processor, Address: containing an address of global memory, Contents: containing a word to be written into or read from global memory, and Program_Counter: containing the line number of the instruction to be executed next. The instructions to be simulated are limited to the following:

READ: Read the word of global memory specified by Address into Contents.

WRITE: Write the Contents register into the global memory location specified by Address.

OP R_a R_b: Perform OP on R_a and R_b leaving the result in R_b. Here OP may be Add, Subtract, or, Shift.

MOVE R_a R_b: Move R_a to R_b.

BLT R L: Branch to line L if the contents of R is less than zero.

$I(n)/\log n$ words of memory contain $\log n$ bits each of the input, or even if all $I(n)$ bits are placed in the first word, then all our results remain unchanged. Note that this is not true of the models used in for example [3]. There processors are assumed to have unlimited power and thus the partition of the inputs is crucial.

It is straightforward to write a first-order inductive definition for the relation VALUE$(\overline{p}, \overline{t}, \overline{x}, r, b)$ meaning that bit $\overline{x}$ in register r of processor $\overline{p}$ just after step $\overline{t}$ is equal to b. Note that since the number of processors, the time, and the word size are all polynomially bounded, a constant number of variables ranging from 0 to $n-1$ suffice to specify each of these values.

The inductive definition of the relation VALUE$(\overline{p}, \overline{t}, \overline{x}, r, b)$ is a disjunction depending on the value of $\overline{p}$'s program counter at time $\overline{t}-1$. The most interesting case is when the instruction to be executed is READ. Here we simply find the most recent time $\overline{t'} < \overline{t}$ at which the word specified by $\overline{p}$'s Address register at time $\overline{t}$ was written into, and the lowest numbered processor $\overline{p'}$ that wrote into this address at time $\overline{t'}$. In this way we can access the answer, namely the $\overline{x}^{\text{th}}$ bit of $\overline{p'}$s Contents register at time $\overline{t'}$.

It remains to check that Addition, Subtaction, BLT, and Shift are first-order expressible, and that we can express the fact that each processor begins with its own processor number in its Processor register. Addition was done in Example 2.1. In a similar way we can express Subtraction, and Less Than. The main place we need the BIT relation is to express the fact that the intitial contents of each processor's Processor register is its processor number. The relation BIT allows us to translate between variable numbers and words in memory. Using BIT we can also express addition on variable numbers and thus express the Shift operation.

Thus we have described an inductive definition of the relation VALUE, coding M's entire computation. Furthermore, one iteration of the definition occurs for each step of M. ∎

Lemma 3.2 *For polynomially bounded, and parallel time constructible $t(n)$,*

$$FO[t(n)] \subseteq CRAM[t(n)]$$

Proof Let the FO$[t(n)]$ problem be determined by the following quantifier free formulas and quantifier block,

$$M_0, M_1, \ldots, M_k, \quad \text{QB} = (Q_1 x_1 . M_1) \ldots (Q_k x_k . M_k) .$$

Our CRAM must test whether an input structure $\mathcal{A}$ satisfies the sentence,

$$\varphi_n \equiv [\text{QB}]^{t(n)} M_0 .$$

The CRAM will use n^k processors and n^{k-1} bits of global memory. Note that each processor has a number $a_1 \ldots a_k$ with $0 \leq a_i < n$. Using the Shift operation it can retrieve each of the a_is in constant time.[6]

The CRAM will evaluate φ_n from right to left, simultaneously for all values of the variables $x_1, \ldots, x_k$. For $0 \leq r \leq t(n) \cdot k$, let,

$$\varphi_n^r \equiv (Q_i x_i . M_i) \ldots (Q_k x_k . M_k)[\mathrm{QB}]^q M_0 ,$$

where $r = k \cdot (q+1) + 1 - i$. Let $x_1 \ldots \hat{x}_i \ldots x_k$ be the $k-1$-tuple resulting from $x_1 \ldots x_k$ by removing x_i. We will now give a program for the CRAM which is broken into rounds each consisting of three processor steps such that:

(*) Just after the r^{th} round, the contents of memory location $a_1 \ldots \hat{a}_i \ldots a_k$ is 1 or 0 according as whether $\mathcal{A} \models \varphi_n^r(a_1, \ldots, a_k)$ or not.

Note that x_i does not occur free in φ_n^r! At the r^{th} round, processor number $a_1 \ldots a_k$ executes the following three instructions according to whether $Q_i = \exists$ or $Q_i = \forall$:

$\{Q_i = \exists\}$

1. $b \leftarrow loc(a_1 \ldots \hat{a}_{i+1} \ldots a_k)$;
2. $loc(a_1 \ldots \hat{a}_i \ldots a_k) \leftarrow 0$;
3. if $M_i(a_1, \ldots, a_k)$ and b then $loc(a_1 \ldots \hat{a}_i \ldots a_k) \leftarrow 1$;

$\{Q_i = \forall\}$

1. $b \leftarrow loc(a_1 \ldots \hat{a}_{i+1} \ldots a_k)$;
2. $loc(a_1 \ldots \hat{a}_i \ldots a_k) \leftarrow 1$;
3. if $M_i(a_1, \ldots, a_k)$ and $\neg b$ then $loc(a_1 \ldots \hat{a}_i \ldots a_k) \leftarrow 0$;

It is not hard to prove by induction that (*) holds, and thus that the CRAM simulates the formula. ∎

Remark 3.3 *The proof of Lemma 3.2 provides a very simple network for simulating a FO[t(n)] property. The network has n^{k-1} bits of global memory and kn^k gates, where k is the number of distinct variables in the quantifier*

[6] This is obvious if n is a power of 2. If not, we can just let each processor break its processor number into k $\lceil \log n \rceil$-tuples of bits. If any of these is greater than or equal to n, then the processor should do nothing during the entire computation.

block. Each gate of the network is connected to two bits of global memory in a simple connection pattern. The blowup of processors going from CRAM to FO to CRAM seems large (cf. Corollary 4.1); however, it is plausible to build first-order networks with billions of processing elements, i.e. gates, thus accommodating fairly large n and moderately large k.

An immediate corollary of Theorem 1.1 is that the complexity class CRAM$[t(n)]$ is not affected by minor changes in how the input is arranged, nor in the global memory word size, nor even by a change in the convention on how write conflicts are resolved.

Corollary 3.4 *For any fucntion $t(n)$, the complexity class CRAM$[t(n)]$ is not changed if we modify the definition of a CRAM in any consistent combination of the following ways. (By consistent we mean that we don't allow input words larger than the global word size, nor larger than the allowable length of applications of Shift.)*

1. *Change the input distribution so that either*
   (a) *The entire input is placed in the first word of global memory.*
   (b) *The $I(n)$ bits of input are placed $\log n$ bits at a time in the first $I(n)/\log n$ words of global memory.*
2. *Change the global memory word size so that either*
   (a) *The global word size is one, i.e. words are single bits. (Local registers do not have this restriction so that the processor's number may be stored and manipulated.)*
   (b) *The global word size is bounded by $O[\log n]$.*
3. *Modify the Shift operation so that shifts are limited to the maximum of the input word size and of the log base two of the number of processors.*
4. *Remove the polynomial bound on the number of memory locations, thus allowing an unbounded global memory.*
5. *Instead of the priority rule for the resolution of write conflicts, adopt the common write rule in which different processors never write different values into the same memory location at a given time step.*

Proof The proof is that Lemmas 3.1 and 3.2 still hold with any consistent set of these modifications. This is immediate for Lemma 3.1. For Lemma 3.2, we must only show that processor number $a_1 \ldots a_k$ still has the power

in constant time to evaluate the quantifier free formula $M_i(a_1, \ldots, a_k)$, and to name the global memory location $a_1 \ldots \hat{a}_i \ldots a_k$, for $1 \leq i \leq k$. Recall that we are assuming that the input structure $\mathcal{A} = \langle \{0, 1, \ldots, n-1\}, R_1^{\mathcal{A}} \ldots R_p^{\mathcal{A}}, c_1^{\mathcal{A}} \ldots c_q^{\mathcal{A}} \rangle$ is coded as a bit string of length $I(n) = n^{r_1} + \cdots + n^{r_p} + q\lceil \log n \rceil$. It is clear that all of the consistent modifications, above, allow processor $a_1 \ldots a_k$ to test in constant time whether or not the relation $R(t_1, \ldots, t_r)$ holds, where R is an input or logical relation, and $t_j \in \{a_1, \ldots, a_k\} \cup \{c_j | 1 \leq j \leq q\}$.

∎

4 On the Efficiency of the Simulations

In this section we analyze the proof of Theorem 1.1 in more detail in order to give the following bounds for translating between CRAM and IND. After we prove Corollary 4.1, we discuss the cost of the simulation, and how these bounds can be improved. The proofs in this section involve counting how many variables are needed in various first-order formulas. This whole section should be omitted by the casual reader.

Corollary 4.1 *Let CRAM$[t(n)]$-PROC$[p(n)]$ be the complexity class CRAM$[t(n)]$ restricted to machines using at most $O[p(n)]$ processors. Let IND$[t(n)]$-VAR$[v(n)]$ be the complexity class IND$[t(n)]$ restricted to inductive definitions using at most $v(n)$ distinct variables. Assume for simplicity that the maximum size of a register word, and $t(n)$ are both $o[\sqrt{n}]$, and that $\pi \geq 1$ is a natural number. Then,*

$$\begin{aligned} &CRAM[t(n)]\text{-}PROC[n^{\pi}] \\ &\qquad \subseteq \quad IND[t(n)]\text{-}VAR[2\pi + 2] \\ &\qquad\qquad \subseteq \quad CRAM[t(n)]\text{-}PROC[n^{2\pi+2}] \end{aligned}$$

Proof We prove these bounds using the following two lemmas.

Lemma 4.2 *If the maximum size of a register word, and $t(n)$ are both $o[\sqrt{n}]$, and if M is a CRAM$[t(n)]$-PROC$[n^{\pi}]$ machine, then the inductive definition of VALUE may be written using $2\pi + 2$ variables.*

Proof We write out the inductive definition of VALUE in enough detail to count the number of variables used:

$$\text{VALUE}(\overline{p}, t, x, r, b) \equiv Z \vee W \vee S \vee R \vee M \vee B \vee A\,.$$

Where the disjuncts have the following intuitive meanings:

Z: $t = 0$ and the initial value of r is correct.

W: $t \neq 0$ and the instruction just executed is WRITE, and the value of r is correct, i.e. unchanged unless r is Program_Counter.

S, R, M, B, A: Similarly for SHIFT, READ, MOVE, BLT, and, ADD or SUBTRACT, respectively.

It suffices to show that each disjunct can be written using the number of variables claimed. First we consider the disjunct, Z. The only interesting part of Z is the case where r is 'Processor'. In this case we use the relation BIT to say that $b = 1$ iff the x^{th} bit of $\overline{p}$ is 1. No extra variables are needed. Note that the number of free variables in the relation is $\pi + 1$ because we may combine the values t, x, r, and b into a single variable.

Next we consider the case of Addition. Recall that the main work is to express the carry bit:

$$C[A, B](x) \equiv (\exists y < x)[A(y) \wedge B(y) \wedge (\forall z.y < z < x)A(z) \vee B(z)]$$

This definition uses two extra variables. Thus $\pi + 3 \leq 2\pi + 2$ variables certainly suffice. The cases S, M, and B are simpler.

The last, and most interesting case is R. Here we must say,

1. The instruction just executed is READ, and,
2. Register r is the Contents register, and,
3. There exists a processor $\overline{p'}$ and a time t' such that:
   (a) $t' < t$, and,
   (b) Address$(\overline{p'}, t')$ =Address$(\overline{p}, t)$, and,
   (c) VALUE$(\overline{p'}, t', x, r, b)$, and,
   (d) Processor $\overline{p'}$ wrote at time t', and,
   (e) For all $\overline{p''} < \overline{p'}$, if $\overline{p''}$ wrote at time t', then Address$(\overline{p''}, t') \neq$Address$(\overline{p'}, t')$, and,
   (f) For all t'' such that $t' < t'' < t$ and for all $\overline{p''}$, if $\overline{p''}$ wrote at time t'', then Address$(\overline{p''}, t'') \neq$Address$(\overline{p'}, t')$.

There is much work to be done. The following general directions suggest themselves:

1. This paper provides a new way to think about parallel programming. The programmer provides efficient inductive definitions of the problem to be solved. Our simulation results then automatically give an efficient implementation on a CRAM. Much work is needed exploring whether or not this approach is practical.

2. We have given characterizations of parallel time and number of processors in terms of the depth and number of variables in inductive definitions. One should now develop upper and lower bounds on these parameters for all sorts of problems. We also feel that the analysis of the simulation in Section 4 can and should be improved.

3. There are many fascinating questions concerning uniformity and the power of precomputation. We hope that the notion of syntactic uniformity of circuits will help researchers determine when precomputation/non-uniformity can help; or, to prove lower bounds on what can be done by uniform circuits and formulas.

Acknowledgements Thanks to Steve Cook, Steven Lindell, Ruben Michel, and Larry Ruzzo who contributed comments and corrections to previous drafts of this paper.

References

[1] David Barrington, "Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1," *18th ACM STOC* (1986), 1-5.

[2] David Mix Barrington, Neil Immerman, and Howard Straubing, "On Uniformity Within NC^1," *Third Annual Structure in Complexity Theory Symp.* (1988), 47-59.

[3] Paul Beame, "Limits on the Power of Concurrent-Write Parallel Machines," *18th ACM STOC* (1986), 169-176.

[4] Ashok Chandra, Larry Stockmeyer and Uzi Vishkin, "Constant Depth Reducibility," *SIAM J. of Comp.* **13**, No. 2 (1984), 423-439.

[5] Steve Cook, "A Taxonomy of Problems with Fast Parallel Algorithms," *Information and Control* **64** (1985), 2-22.

[6] Herbert Enderton, *A Mathematical Introduction to Logic,* Academic Press (1972).

[7] Ron Fagin, "Generalized First-Order Spectra and Polynomial-Time Recognizable Sets," in *Complexity of Computation,* (ed. R. Karp), *SIAM-AMS Proc.* **7** (1974), 27-41.

[8] Faith Fich, Friedhelm Meyer auf der Heide, Prabhakar Ragde, and Avi Wigderson, "One, Two, Three, . . ., Infinity: Lower bounds for Parallel Computation," *17th ACM STOC Symp.* (1985), 48-58.

[9] Faith Fich, Prabhakar Ragde, and Avi Wigderson, "Relations Between Concurrent-Write Models of Parallel Computation," *Third ACM Symp. on Principles of Distributed Computing* (1984), 179-189.

[10] Merrick Furst, James Saxe, and Mike Sipser, "Parity, Circuits, and the Polynomial-Time Hierarchy," *22nd IEEE FOCS Symp.*(1981), 260-270.

[11] David Harel and Dexter Kozen, "A Programming Language for the Inductive Sets, and Applications," *Ninth ICALP, Springer-Verlag Lecture Notes in Computer Science 140* (1982).

[12] Johan Hastad, "Almost Optimal Lower Bounds for Small Depth Circuits," *18th ACM STOC Symp.* (1986), 6-20.

[13] Hong Jia-wei, "On Some Deterministic Space Complexity Problems," *SIAM J. Comput.* **11** (1982), 591-601.

[14] Neil Immerman, "Number of Quantifiers is Better than Number of Tape Cells," *JCSS* **22**, No. 3 (1981), 65-72.

[15] Neil Immerman, "Upper and Lower Bounds for First Order Expressibility," *JCSS* **25**, No. 1 (1982), 76-98.

[16] N. Immerman, "Relational Queries Computable in Polynomial Time," *Information and Control,* 68 (1986), 86-104. A preliminary version of this paper appeared in *14th ACM STOC Symp.* (1982), 147-152.

[17] N. Immerman, "Languages That Capture Complexity Classes," *SIAM J. Comput.* **16**, No. 4 (1987), 760-778. A preliminary version of this paper appeared in *15th ACM STOC Symp.* (1983), 347-354.

[18] Neil Immerman, "Expressibility as a Complexity Measure: Results and Directions," *Second Structure in Complexity Theory Conf.* (1987), 194-202.

[19] N. Immerman, "Descriptive and Computational Complexity," to appear in *Proc. AMS Short Course in Computational Complexity Theory* (1988).

[20] Yiannis N. Moschovakis, *Elementary Induction on Abstract Structures,* North Holland (1974).

[21] Larry Ruzzo, "On Uniform Circuit Complexity," *J. Comp. Sys. Sci.,* **21**, No. 2 (1981), 365-383.

[22] Larry Stockmeyer, "The Polynomial-Time Hierarchy," *Theoretical Comp. Sci.* **3** (1977), 1-22.

[23] Larry Stockmeyer and Uzi Vishkin, "Simulation of Parallel Random Access Machines by Circuits," *SIAM J. of Comp.* **13**, No. 2 (1984), 409-422.

[24] Andrew Chi-Chih Yao, "Separating the Polynomial-Time Hierarchy by Oracles," *26th IEEE Symp. on Foundations of Comp. Sci.* (1985), 1-10.