Descriptive and Computational Complexity*

Neil Immerman†

Computer Science Department
University of Massachusetts
Amherst, MA 01003

1 Introduction

Computational complexity began with the natural physical notions of time and space. Given a property, S, an important issue is the computational complexity of checking whether or not an input satisfies S. For a long time, the notion of complexity referred to the time or space used in the computation. A mathematician might ask, "What is the complexity of *expressing* the property S?" It should not be surprising that these two questions – that of checking and that of expressing – are related. However it is startling how closely tied they are when the second question refers to expressing the property in first-order logic. Many complexity classes originally defined in terms of time or space resources have precise definitions as classes in first-order logic.

In 1974 Fagin gave a characterization of nondeterministic polynomial time (NP) as the set of properties expressible in second-order existential logic. We will begin with this result and then survey some more recent work relating first-order expressibility to computational complexity. Some of the results arising from this approach include characterizing polynomial time (P) as the set of properties expressible in first-order logic plus a least fixed point operator (to be defined later), and showing that the set of first-order inductive definitions for finite structures is closed under complementation.

*Lecture Notes for the AMS Short Course in Computational Complexity Theory, Jan. 5-6, 1988, Atlanta, GA. Appeard in Proceedings of Symposia in Applied Mathematics, Vol. 38 (1989), 75-91.

†Research supported by NSF Grant DCR-8603346.

Recently our technology has become able to build highly parallel computers containing thousands of processors working simultaneously. For this reason the theory of parallel computation, and the study of parallel time as a computational resource has become an important area. We will discuss parallel computation, and then show that parallel time can be neatly characterized in terms of first-order expressibility: The minimum parallel time needed to compute a property using at most polynomially many processors is equal to the minimum depth of a first-order inductive definition of the property.

An apparently weaker operator than full inductive definitions is the power to take the reflexive, transitive closure (TC) of any defined binary relation $\varphi(\bar{x}, \bar{y})$. We show that (FO + pos TC), the class of properties expressible using TC positively (i.e. not within any negations) is equal to NSPACE[$\log n$]. Finally we show the very surprising, recent result that (FO + pos TC) is closed under complementation. A corollary of this result is:

Theorem For any $s(n) \geq \log n$, nondeterministic space $s(n)$ is closed under complementation.

2 Some Logical Definitions

We begin with some precise definitions. The reader is referred to [7] for background in first-order logic.

A *vocabulary* $\tau = \langle R_1^{a_1} \ldots R_k^{a_k}, c_1 \ldots c_r \rangle$ is a tuple of relation symbols and constant symbols. $R_i^{a_i}$ is a relation symbol of arity a_i. In the sequel we will often omit the superscripts to improve readability. A finite *structure* with vocabulary τ is a tuple, $\mathcal{A} = \langle \{0, 1, \ldots, n-1\}, R_1^{\mathcal{A}} \ldots R_k^{\mathcal{A}}, c_1^{\mathcal{A}} \ldots c_r^{\mathcal{A}} \rangle$, consisting of a universe $U^{\mathcal{A}} = \{0, \ldots, n-1\}$ and relations $R_1^{\mathcal{A}} \ldots R_k^{\mathcal{A}}$ of arities $a_1, \ldots, a_k$ on $U^{\mathcal{A}}$ corresponding to the relation symbols $R_1^{a_1} \ldots R_k^{a_k}$ of τ, and constants $c_1^{\mathcal{A}} \ldots c_r^{\mathcal{A}}$ from $U^{\mathcal{A}}$ corresponding to the constant symbols $c_1 \ldots c_r$ from τ. We write $|\mathcal{A}|$ to denote n, the cardinality of the universe of $\mathcal{A}$.

For example, if τ_g consists of a single binary relation symbol E (standing for edge) then a structure $G = \langle \{0 \ldots n-1\}, E \rangle$ with vocabulary τ_g is a graph on n vertices. Similarly if τ_s consists of a single unary relation symbol M then a structure $S = \langle \{0 \ldots n-1\}, M \rangle$ with vocabulary τ_s is a binary string of length n.

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. For convenience we also include the constant symbols 0 and *max* refering to the first and last elements of the structure respectively, and the logical relation $s(x, y)$ true when x is the immediate successor of y in the $\leq$ ordering. For technical reasons we also include the logical relation BIT, where $\mathrm{BIT}(x, y)$ holds iff the xth 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 symbols and constant symbols: $=, \leq, s, \mathrm{BIT}, 0, max$, using logical connectives: $\wedge, \vee, \neg$, variables: $x, y, z, ...$, and quantifiers: $\forall, \exists$.

From now on, we will think of a *problem* as a set of structures of some vocabulary τ. Thus P, NP, etc. will be the set of problems in P (polynomial time), NP (non-deterministic polynomial time), etc. 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. Define FO to be the set of all first-order expressible problems.

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).

It is straightforward to check that FO is contained in DSPACE[$\log n$].[3] We will see in Section 5 that FO is equal to the set of problems checkable

[1] Of course some of these logical relations are redundant. We include all of them to make the statements of some of our theorems simpler: Theorem 4.5 requires the constant symbols. Theorem 5.1 requires the relation BIT when $t(n)$ is $o(\log n)$. Theorem 6.2 requires the successor relation and the constant symbols.

[2] This is a standard construction, see e.g. [29].

[3] To see this suppose we are given a first-order sentence

$$\varphi \;\equiv\; (\exists x_1)(\forall x_2)\ldots(Q_k x_k)M(\bar{x})\ .$$

in constant time on a concurrent parallel random access machine (CRAM). This is a very weak complexity class. In the next several sections we will discuss strengthenings of first-order logic to languages that capture more important complexity classes.

3 Second-Order Logic and Fagin's Theorem

In second-order logic we have first-order logic, plus new relation variables over which we may quantify. Let A_i^j be a j-ary relation variable. Then $(\forall A_i^j)\varphi$ means that for all choices of j-ary relation A_i^j, φ holds. It is well known that second-order formulas may be transformed into prenex form, with all second-order quantifiers in front. Let SO be the set of second-order expressible properties, and let (SO $\exists$) be the set of second-order properties that may be written in prenex form with no universal second-order quantifiers. Consider the following example, in which R, Y, and B are unary relation variables,

$$\alpha \equiv (\exists R)(\exists Y)(\exists B)(\forall x)\Big[(R(x) \vee Y(x) \vee B(x)) \wedge (\forall y)\Big(E(x,y) \rightarrow \neg(R(x) \wedge R(y)) \wedge \neg(Y(x) \wedge Y(y)) \wedge \neg(B(x) \wedge B(y))\Big)\Big]$$

Observe that a graph G satisfies α iff G is 3-colorable.[4] The formula α is an example of the following theorem of Fagin,

Theorem 3.1 *[8] (SO $\exists$) = NP .*

Proof ($\subseteq$): Given a second-order existential sentence $\Phi \equiv (\exists R_1^{a_1}) \ldots (\exists R_k^{a_k})\varphi$ the task of our NP machine is to test whether its input $\mathcal{A}$ satisfies Φ. To do this the machine can guess the relations $R_1^{a_1}, \ldots, R_k^{a_k}$. The relation $R_i^{a_i}$ may be specified by a string of n^{a_i} bits where $n = |\mathcal{A}|$. The task of testing whether $\mathcal{A}, R_1^{a_1}, \ldots, R_k^{a_k} \models \varphi$ is a problem in FO and thus certainly in P.

($\supseteq$): Conversely, let N be a nondeterministic Turing machine that uses time n^k for inputs $\mathcal{A}$ with $n = |\mathcal{A}|$. We will write a second-order sentence

A logspace Turing machine T must check whether its input $\mathcal{A}$ satisfies φ. Let $n = |\mathcal{A}|$. T marks off $k \log n$ tape cells in which it can systematically cycle through all values of $x_1, \ldots, x_k$. For each value of $\bar{x}$, T tests whether or not the quantifier free matrix M holds in $\mathcal{A}$.

[4] A graph is 3-colorable iff its vertices may be colored with one of three colors so that no two adjacent vertices are the same color. Three colorabliity is an NP complete property.

$\Phi \equiv (\exists C_1 \ldots C_s)\varphi$ that says, "There exists an accepting computation $\bar{C}$ of N." More precisely, the first-order sentence φ will have the property that $\mathcal{A}, \bar{C} \models \varphi$ iff $\bar{C}$ is an accepting computation of N on input $\mathcal{A}$. Thus,

$$\mathcal{A} \models \Phi \;\Leftrightarrow\; N \text{ accepts } \mathcal{A}$$

We now describe how to code N's computation. $\bar{C}$ consists of a matrix $\bar{C}(\bar{s}, \bar{t})$ of n^{2k} tape cells with space $\bar{s}$ and time $\bar{t}$ varying between 0 and $n^k - 1$. We use a k-tuple of variables $\bar{t} = t_k, \ldots, t_1$ and $\bar{s} = s_k, \ldots s_1$ each ranging over the universe of $\mathcal{A}$, i.e. from 0 to $n-1$, to code these values. For each $\bar{s}, \bar{t}$ pair, $\bar{C}(\bar{s}, \bar{t})$ codes the tape symbol σ that appears in the $\bar{s}^{\text{th}}$ cell at time $\bar{t}$, if n's head is not on this cell. If the head is present then $\bar{C}(\bar{s}, \bar{t})$ codes the pair $\langle q, \sigma \rangle$ consisting of N's state q at time $\bar{t}$, and the tape symbol σ. Let $R = \{r_1, \ldots, r_s\} = (Q \times \Sigma) \cup \Sigma$ be a listing of the possible contents of a computation cell. We will let C_i be a $2k$-ary relation variable for $1 \leq i \leq s$. The intuitive meaning of $C_i(\bar{s}, \bar{t})$ is that the computation cell $\bar{s}$ at time $\bar{t}$ contains symbol r_i.

It is now fairly straightforward to write the first-order sentence $\varphi(\bar{C})$ saying that $\bar{C}$ codes a valid accepting computation of N. Note that the input $\mathcal{A}$ is coded in the contents of N's tape at time 0. One can code $\mathcal{A}$ in a sequence of $n^{a_1} + n^{a_2} + \ldots + n^{a_t} + r \log n$ bits where $\mathcal{A}$ has vocabulary $\tau = \langle R_1^{a_1}, \ldots, R_t^{a_t}, c_1, \ldots, c_r \rangle$. The sentence φ must assert that the input is coded correctly, e.g., φ includes the following clause meaning that cell $0 \ldots 0 s_1 \ldots s_{a_1}$ is a one iff $R_1^{a_1}(s_1, \ldots, s_{a_1})$ holds.

$$(s_k = 0 \wedge s_{k-1} = 0 \wedge \ldots \wedge s_{a_1+1} = 0) \;\rightarrow\; (C_1(\bar{s}) \leftrightarrow R_1^{a_1}(s_1, \ldots s_{a_1}))$$

The sentence φ must also assert that the contents of tape cell $(\bar{s}, \bar{t}+1)$ follows from the contents of cells $(\bar{s}-1, \bar{t})$, $(\bar{s}, \bar{t})$, and $(\bar{s}+1, \bar{t})$ via a move of N. Finally, φ says that an accept state is eventually reached. ∎

The following corollary due to Stockmeyer gives a nice characterization of the polynomial-time hierarchy.

Corollary 3.2 *[28] PH = SO .*

4 Inductive Definitions

A useful way to increase the power of first-order logic without jumping all the way up to second order logic is to add the power to define new relations

by induction. For example, consider the vocabulary $\tau_g = \langle E \rangle$ of graphs. We can define the reflexive, transitive closure E^* of E as follows. Let R be a binary relation variable and consider the formula,

$$\varphi(R, x, y) \equiv x = y \vee \exists z(E(x, z) \wedge R(z, y)) \tag{1}$$

The formula φ formalizes an inductive definition of E^* which may be more suggestively written as follows,

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

For any structure $\mathcal{A}$ with vocabulary τ_g, φ induces a map from binary relations on the universe of $\mathcal{A}$ to binary relations on the universe of $\mathcal{A}$,

$$\varphi_{\mathcal{A}}(R) = \{\langle a, b \rangle \mid \mathcal{A} \models \varphi(R, a, b)\}$$

Note that since R appears only positively in φ, $\varphi_{\mathcal{A}}$ is monotonic. Let $\varphi^r_{\mathcal{A}}$ denote $\varphi_{\mathcal{A}}$ iterated r times. With φ defined as in Equation 1, $\mathcal{A}$ any graph, and $r \geq 0$ observe that,

$$\varphi^r_{\mathcal{A}}(\emptyset) = \{\langle a, b \rangle \in (U^{\mathcal{A}})^2 \mid \text{distance}(a, b) \leq r - 1\} .$$

Thus, in particular, if $n = |\mathcal{A}|$, then $\varphi^n_{\mathcal{A}}(\emptyset) = E^* =$ the least fixed point of $\varphi_{\mathcal{A}}$. In general, let $\psi(R, x_1, \ldots, x_k)$ be an R-positive first-order formula, i.e., R does not occur within any negation signs. Then for any finite structure $\mathcal{A}$, the least fixed point of $\psi_{\mathcal{A}}$ exists and is equal to $\psi^r_{\mathcal{A}}(\emptyset)$ where r is minimal so that $\psi^r_{\mathcal{A}}(\emptyset) = \psi^{r+1}_{\mathcal{A}}(\emptyset)$. The number r is called the *closure ordinal* of ψ in $\mathcal{A}$ and denoted $|\psi_{\mathcal{A}}|$. Note that $|\psi_{\mathcal{A}}| \leq n^k$ where $n = |\mathcal{A}|$. This is true because each application of $\psi_{\mathcal{A}}$ before the $r + 1^{\text{st}}$ adds some new k-tuple to the relation. We also define the closure ordinal of ψ by $|\psi|(n) = max\{|\psi_{\mathcal{A}}| \mid n = |\mathcal{A}|\}$.

Remark 4.1 *For φ given in Equation 1 the closure ordinal $|\varphi|(n) = n$. However, the following alternate inductive definition of E^* has closure ordinal $|\beta|(n) = \lceil \log n \rceil + 1$.*

$$\beta(R, x, y) \equiv x = y \vee E(x, y) \vee \exists z(R(x, z) \wedge R(z, y))$$

Definition 4.2 *We now define (FO + LFP) to be the set of first-order inductive definitions. We do this by adding a least fixed point operator (LFP) to first-order logic. If $\varphi(R^k, x_1, \ldots, x_k)$ is an R^k-positive formula in (FO +*

LFP), then $(\mathrm{LFP}_{R^k x_1 \ldots x_k}\, \varphi)$ *is a formula in (FO + LFP) denoting the least fixed point of* φ*. We also define IND[*$f(n)$*] to be the sublanguage of (FO + LFP) in which we only include least fixed points of first-order formulas* φ *for which* $|\varphi|$ *is* $O[f(n)]$*. For example, the reflexive, transitive closure of E is expressible as* $(\mathrm{LFP}_{Rxy}\, \beta)$ *and is thus in IND[*$\log n$*]. Note also that,*

$$(FO + LFP) = \bigcup_{k=1}^{\infty} IND[n^k] .$$

Immerman and Vardi independently characterized the complexity of (FO + LFP) as follows,

Theorem 4.3 *[15, 31] (FO + LFP) = P .*

Proof ($\subseteq$): By the above discussion, for an input $\mathcal{A}$ of size n, $(\mathrm{LFP}_{Rx_1 \ldots x_k}\, \varphi) = \varphi_{\mathcal{A}}^{n^k}$. Thus we need only evaluate the formula φ at most n^k times.

($\supseteq$): Let M be a deterministic Turing machine that runs in time n^k for input structures of size n. Recall the proof of Theorem 3.1 where we existentially quantified $\bar{C}$, an accepting computation of the nondeterministic Turing machine N. Here we define *the* computation $\bar{C}$ of M by induction, and assert that it ends in an accepting state. Instead of presenting the details now, we defer them until the proof of Theorem 5.1 for which the present theorem is a corollary. ∎

In the proof of Theorem 3.1 we did not need to assume that the logical relation $\leq$ is present. This is because in (SO $\exists$) we can existentially quantify a binary relation L and assert that it is a total ordering on the universe. However, having the ordering is crucial to the truth of Theorem 4.3. For example, even the trivial graph property of having an even number of edges is not expressible in (FO(wo$\leq$) + LFP)[5] [3, 17]. Consider the language (FO(wo$\leq$) + pos LFP), in which LFP does not occur within any negation symbols. An interesting question is whether or not this class is closed under complementation. In the case of infinite structures the answer is no [26]. For finite structures, Chandra and Harel [3] conjectured that the answer is no.

For finite ordered structures, the proof of Theorem 4.3 shows that one fixed point is enough to define a whole polynomial time computation and

[5] When we say, "wo$\leq$," we mean without any of the logical representations of ordering, i.e. $\leq, s, \mathrm{BIT}$.

thus (FO + pos LFP) = P = (FO + LFP). It was surprising to find that even without ordering fixed points are closed under complementation.

Theorem 4.4 *[15] (FO(wo≤) + pos LFP) = (FO(wo≤) + LFP) .*

Proof Suppose we are given an R-positive first-order formula $\varphi(R, x_1, \ldots, x_k)$. We must show that the relation $\neg(\mathrm{LFP}_{R\bar{x}}\,\varphi)$ is expressible in the form $(\mathrm{LFP}_{S y_1 \ldots y_t}\,\psi)$ for some S-positive first-order formula ψ. We prove this using the Stage Comparison Theorem of Moschovakis [26]. Fix a finite structure $\mathcal{A}$. Define the relations $<_\varphi$ and $\leq_\varphi$ on the set of k-tuples from the universe of $\mathcal{A}$ as follows. Let $\mathrm{cl}(\varphi_\mathcal{A}, \bar{a})$ be the minimum r such that $\bar{a} \in \varphi^r_\mathcal{A}(\emptyset)$ if $\bar{a} \in \mathrm{LFP}(\varphi_\mathcal{A})$ and ∞ otherwise. We define $\bar{a} \leq_\varphi \bar{b}$ (resp. $\bar{a} <_\varphi \bar{b}$) to mean that $\mathrm{cl}(\varphi_\mathcal{A}, \bar{a}) < \infty$ and $\mathrm{cl}(\varphi_\mathcal{A}, \bar{a}) \leq \mathrm{cl}(\varphi_\mathcal{A}, \bar{b})$ (resp. $\mathrm{cl}(\varphi_\mathcal{A}, \bar{a}) < \mathrm{cl}(\varphi_\mathcal{A}, \bar{b})$). In words, $\bar{a} \leq_\varphi \bar{b}$ if $\bar{a}$ is in the LFP of φ and comes in no later than $\bar{b}$. The stage comparison theorem states that $<_\varphi$ and $\leq_\varphi$ are expressible as a single least fixed point.

Now we have the tools to express the negation of the fixed point of φ. First, it is easy to express the relation $\bar{a} \ll_\varphi \bar{b}$ meaning that $\mathrm{cl}(\varphi_\mathcal{A}, \bar{a}) < \mathrm{cl}(\varphi_\mathcal{A}, \bar{b}) - 1$. Of course the closure ordinal $|\varphi_\mathcal{A}|$ is finite since $\mathcal{A}$ is finite. Using $\leq_\varphi$ and $\ll_\varphi$ we can express the fact that some $\bar{m}$ has this maximal closure ordinal,

$$\mathrm{MAX}(\bar{m}) \;\equiv\; (\bar{m} \leq_\varphi \bar{m}) \wedge (\forall \bar{x})(\bar{x} \leq_\varphi \bar{m} \vee\; \bar{m} \ll_\varphi \bar{x}) \;.$$

Using MAX we can then express negation as follows:

$$\neg(\mathrm{LFP}_{R x_1 \ldots x_k}\,\varphi)(\bar{y}) \;\equiv\; (\exists \bar{m})(\mathrm{MAX}(\bar{m}) \wedge \bar{m} <_\varphi \bar{y}) \;.$$

It is well known [26, 15] that the two positive fixed points for $\leq_\varphi$ and $<_\varphi$, plus the finitely many extra quantifiers can be merged into a single positive fixed point. ∎

To conclude this section we note that the above results lead to the following normal form theorem for the language (FO + LFP).

Theorem 4.5 *[15] Let φ be any formula in the language (FO + LFP). Whether or not the ordering relations are present, there exists a first-order formula ψ such that*

$$\varphi \;\equiv\; (\mathrm{LFP}\,\psi)(\bar{0})$$

5 Inductive Depth Equals Parallel Time

In this section we study the relationship between first-order expressibility and parallel complexity. First we precisely define the class CRAM-TIME$[t(n)]$, which is intuitively the set of problems checkable by an idealized parallel computer in time $t(n)$.

The concurrent random access machine (CRAM) is essentially the concurrent read, concurrent write parallel random access machine (CRCW PRAM) described in [29]. 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. 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 [29] 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 a polynomial amount of hardware (i.e. polynomially many processors and polynomially many bits of memory) and time $O[t(n)]$. The input to a CRAM is a binary string coding a first-order structure $\mathcal{A}$ of vocabulary $\tau = \langle R_1^{a_1}, \ldots, R_t^{a_t}, c_1, \ldots, c_r \rangle$. Recall that $\mathcal{A}$ may be coded in a sequence of $m = n^{a_1} + n^{a_2} + \ldots + n^{a_t} + r \log n$ bits. The input string is placed one bit at a time in the first m global memory locations.[6]

The following theorem says that the parallel time needed to check if an input has a certain property S is linearly related to the inductive depth needed to express S.

Theorem 5.1 *[18]For all polynomially bounded $t(n)$,*

$$\text{CRAM}[t(n)] \;=\; \mathit{IND}[t(n)]\,.$$

[6] If placement of the input is varied, e.g. if the first $m/\log n$ words of memory contain $\log m$ bits each of the input, or even if the whole m-bit string is placed in the first memory location, then all our results remain unchanged.

Lemma 5.2 *For any polynomially bounded $t(n)$ we have,*

$$\mathrm{CRAM}[t(n)] \subseteq \mathrm{IND}[t(n)]$$

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.

It is straightforward to write a first-order inductive definition for the relation $\mathrm{VALUE}(\overline{p}, \overline{t}, \overline{x}, r, b)$ meaning that bit $\overline{x}$ in register r of processor $\overline{p}$ at 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 $\mathrm{VALUE}(\overline{p}, \overline{t}, \overline{x}, r, b)$ 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 at time $\overline{t}-1$ is READ. Here we

simply find the most recent time $\overline{t'} < \overline{t} - 1$ at which the word specified by $\overline{p}$'s Address register at time $\overline{t} - 1$ 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 and Subtraction and Less Than are similar. The main place we need the BIT relation is to express the fact that the intitial contents of each processor's Processor register is it's 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. ∎

In order to compute inductive definitions on CRAM's it is convenient to put the inductive definitions into a simple normal form. The following has a straightforward inductive proof, cf. [26, 15].

Fact 5.3 *Let φ be an R-positive first-order formula. Then φ can be written in the following form,*

$$\varphi(R, x_1, \ldots, x_k) \equiv (Q_1 z_1.M_1) \ldots (Q_s z_s.M_s)(\exists x_1 \ldots x_k.M_{s+1}) R(x_1, \ldots, x_k) .$$

where the M_i's are quantifier-free formulas in which R does not occur.

Here the notation $(\forall x.M)\psi$ means $(\forall x)M \rightarrow \psi$, and $(\exists x.M)\psi$ means $(\exists x)M \wedge \psi$. Note that the above requantification of the x_i's means that these variables may occur free in $M_1 \ldots M_s$, but they are bound in M_{s+1} and $R(x_1, \ldots, x_k)$. Note that the same variables may now be requantified. Let us write QB to denote the quantifier block $(Q_1 z_1.M_1) \ldots (Q_s z_s.M_s)(\exists x_1 \ldots x_k.M_{s+1})$. Thus, in particular, for any structure $\mathcal{A}$,and any $r \in \mathbf{N}$,

$$\mathcal{A} \models (\varphi^r_{\mathcal{A}}(\emptyset)) \leftrightarrow ([\text{QB}]^r \mathit{false}) .$$

Here $[\text{QB}]^r$ means QB repeated r times (literally). It follows immediately that if $t = |\varphi|(n)$, and $\mathcal{A}$ is any structure of size n then

$$\mathcal{A} \models (\text{LFP}\, \varphi) \leftrightarrow ([\text{QB}]^t \mathit{false}) .$$

This is the simple form for inductive definitions we wanted. We next show how to evaluate such definitions using a CRAM.

Lemma 5.4 *For polynomially bounded $t(n)$,*

$$\mathrm{IND}[t(n)] \subseteq \mathrm{CRAM}[t(n)]$$

Proof As in the above discussion, let the $\mathrm{IND}[t(n)]$ problem be determined by the following quantifier free formulas and quantifier block,

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

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

$$\varphi_n \equiv [\mathrm{QB}]^{t(n)} \mathit{false} \; .$$

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_i s in constant time.

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,

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

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

$\{Q_i = \forall\}$

1. $b \leftarrow \mathit{loc}(a_1 \ldots \hat{a}_{i+1} \ldots a_k)$;
2. $\mathit{loc}(a_1 \ldots \hat{a}_i \ldots a_k) \leftarrow 1$;

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. It is also straightforward to check for progress after each iteration of the whole quantifier block, and to halt when no such progress occurs. ∎

Theorem 5.1 tells us that inductive depth is exactly equal to parallel time, in the whole range in which inductive depth is defined. If we want to talk about super polynomial parallel time, then we must talk about iterating first-order formulas as in the discussion after Fact 5.3. We now define FO$[t(n)]$ to be the set of properties defined by quantifier blocks iterated $t(n)$ times:

Definition 5.5 *A set C of structures of vocabulary τ is a member of FO$[t(n)]$ iff there exist quantifier free formulas M_i, $0 \leq i \leq k$, from $\mathcal{L}(\tau)$, and a quantifier block,*

$$\text{QB} \;=\; [(Q_1 x_1.M_1) \ldots (Q_k x_k.M_k)]$$

such that if we let $\varphi_n = [\text{QB}]^{t(n)} M_0$, for $n = 1, 2, \ldots$, then for all structures $\mathcal{A}$ of vocabulary τ with $|\mathcal{A}| = n$,

$$\mathcal{A} \in C \;\Leftrightarrow\; \mathcal{A} \models \varphi_n \,.$$

As a corollary to Fact 5.3, and a simple generalization of Theorem 5.1, we obtain the following three results. Note that we need the uniformity assumption on $t(n)$ because unlike a FO$[t(n)]$ property, an inductive definition or a CRAM program must figure out on its own when to stop. To prove Corollary 5.8 just observe that since we defined our CRAM time to be on a machine with polynomially bounded hardware, if such a CRAM is allowed to run for an unlimited amount of time, it can make use of at most exponential time and it can compute exactly the polynomial space properties.

Corollary 5.6 *[18] For all polynomially bounded, parallel time constructible $t(n)$,*

$$\text{FO}[t(n)] \;=\; \text{IND}[t(n)] \,.$$

Corollary 5.7 *[18] For all parallel time constructible $t(n)$,*

$$\text{CRAM}[t(n)] \;=\; \text{FO}[t(n)] \,.$$

Corollary 5.8 *[14]*

$$PSPACE \;=\; \bigcup_{k=1}^{\infty} \text{FO}[2^{n^k}]$$

6 First-Order Logic Plus Transitive Closure

The reflexive, transitive closure is a particularly important case of an inductive definition. Let $\varphi(x_1, \ldots, x_k, x'_1 \ldots x'_k)$ be a formula with $2k$ free variables. We will write $(\mathrm{TC}_{x_1 \ldots x_k x'_1 \ldots x'_k}\ \varphi)$ to denote the reflexive, transitive closure of the binary relation $\varphi(\bar{x}, \bar{x}')$. Let (FO + TC) be the closure of first-order logic with arbitrary occurrences of TC, and let (FO + pos TC) be the restriction of (FO + TC) in which TC never occurs within a negation.

Theorem 6.1 *[16] (FO + pos TC) = NSPACE[$\log n$] .*

Proof ($\subseteq$): The set of relations computable in NSPACE[$\log n$] is closed under first-order quantifiers, $(\forall x)$ and $(\exists x)$ because with space $\log n$ we can cycle through all the values of x. Thus it suffices to show that if $\varphi(\bar{x}, \bar{x}')$ is computable in NSPACE[$\log n$], then so is $(\mathrm{TC}_{\bar{x},\bar{x}'}\ \varphi)$. We can test if the structure $\mathcal{A}$ satisfies $(\mathrm{TC}_{\bar{x},\bar{x}'}\ \varphi)(\bar{a}, \bar{a}')$ as follows: Guess $\bar{b}$ and check that $\mathcal{A} \models \varphi(\bar{a}, \bar{b})$. Next throw away $\bar{a}$ and guess $\bar{c}$ such that $\mathcal{A} \models \varphi(\bar{b}, \bar{c})$. Repeat this process until we guess $\bar{z}$ such that $\mathcal{A} \models \varphi(\bar{y}, \bar{z})$, and $\bar{z} = \bar{a}'$, in which case we accept. The space needed is $3k \log n$ plus the space to check if $\varphi(\bar{x}, \bar{x}')$ holds, where k is the arity of $\bar{x}$.

($\supseteq$): Here we are given an NSPACE[$\log n$] machine N and we must write the sentence $\psi \in$ (FO + pos TC) such that for any structure $\mathcal{A}$,

$$(\mathcal{A} \models \psi) \ \Leftrightarrow\ (N \text{ accepts } \mathcal{A}) \ .$$

Assume for the sake of simplicity that N accepts a graph problem, and uses $k \log n$ bits of work tape. Then any configuration of N can be coded with $k + 3$ variables: $r_1, r_2, w_1, w_2, \ldots, w_k, q$ where $w_1, w_2, \ldots, w_k$ code the work tape, q codes n's state and the position of its work head, and r_1, r_2 code the position of N's read head. Note that the read head is looking at bit $\langle r_1, r_2 \rangle$ of $\mathcal{A}$'s adjacency matrix. Thus the read head is reading a one (resp. a zero) iff $\mathcal{A} \models E(r_1, r_2)$ (resp. $\mathcal{A} \models \neg E(r_1, r_2)$). It is straightforward to see that the predicates $\mathrm{START}(\bar{c})$, $\mathrm{ACCEPT}(\bar{d})$, and $\mathrm{MOVE}(\bar{e}, \bar{f})$, meaning that $\bar{c}$ is the initial configuration of N, $\bar{d}$ is an accept configuration, and that $\langle \bar{e}, \bar{f} \rangle$ is a legal move of N, are all first-order expressible.[7] Thus the

[7] In order to write $\mathrm{MOVE}(\bar{e}, \bar{f})$, we make use of the relation BIT to read the appropriate bit of the variables coding N's work tape. If BIT were not given to us, however, it would still be expressible from $\leq$ using TC. See [16] for this and all the other details of this proof.

sentence we need is

$$\psi \equiv (\exists \bar{c}.\mathrm{START}(\bar{c}))(\exists \bar{d}.\mathrm{ACCEPT}(\bar{d}))(\mathrm{TC}_{\bar{e},\bar{f}}\,\mathrm{MOVE})(\bar{c},\bar{d}) .$$

∎

It is interesting that every property in NSPACE[$\log n$] may be written as a transitive closure of a quantifier free first-order relation. See [16] for the proof of the following normal form theorem.

Theorem 6.2 *Every formula in (FO + pos TC) may be written in the form*

$$(\mathrm{TC}_{\bar{x},\bar{x}'}\,\varphi)(\bar{0},\overline{max})$$

where φ is first-order and quantifier free.

We close with a very surprising result.

Theorem 6.3 *[19] (FO + pos TC) = (FO + TC).*

Proof It will suffice to show that the relation $\neg(\mathrm{TC}_{u,u'}\,E(u,u'))(0,x)$ meaning that there is no path from 0 to x is expressible in (FO + pos TC). We will do this in two lemmas. Let $\mathrm{PATH}(y,d)$ mean that there is a path of length at most d from 0 to y. Obviously this relation is expressible in (FO + pos TC). Suppose that we are given c_d, the exact number of vertices $y \neq 0$ such that $\mathrm{PATH}(y,d)$ holds.[8] Lemma 6.4 shows that in this case the negation of $\mathrm{PATH}(y,d)$ is also expressible in (FO + pos TC). We will then use this result in Lemma 6.5 to show that given c_d we can compute c_{d+1}. It then follows using one more transitive closure, that we can compute c_{n-1}. Thus by Lemma 6.4 again, we can say, "There is no path from 0 to x."

Lemma 6.4 *Let* $\mathrm{NOPAT}H(y,d,c)$ *be a formula such that*

$$\mathrm{NOPAT}H(y,d,c_d) \Leftrightarrow \neg\mathrm{PATH}(y,d) .$$

That is, if $c = c_d$, then $\mathrm{NOPAT}H(y,d,c)$ *is the negation of* $\mathrm{PATH}(y,d)$*; whereas, if $c \neq c_d$ then we don't care what* $\mathrm{NOPAT}H(y,d,c)$ *means. Then such a formula* $\mathrm{NOPAT}H(y,d,c)$ *is expressible in (FO + pos TC).*

[8] We don't include 0 here just because variables range from 0 to $n-1$, while, if we did include 0, then counts would range from 1 to n.

Proof The idea is that if $c = c_d$, then $\neg\text{PATH}(y, d)$ holds iff there exist at least c nonzero vertices $z \neq y$ such that $\text{PATH}(z, d)$. The latter condition is expressible in (FO + pos TC) as follows. First define an α edge between pairs of vertices:

$$\alpha(u, k, u', k') \equiv s(u, u') \wedge \Big[\Big(k = k'\Big) \vee \Big(s(k, k') \wedge \text{PATH}(u', d) \wedge u' \neq y\Big)\Big]$$

Here s is the successor relation. An α edge exists from $\langle u, k\rangle$ to $\langle u+1, k\rangle$ in all cases, and to $\langle u+1, k+1\rangle$ just if $\text{PATH}(u+1, d)$ holds, and $u+1 \neq y$. It follows that the formula

$$\text{NOPATH}(y, d, c) \equiv (y \neq 0) \wedge (\text{TC}_{uk,u'k'}\ \alpha)(0, 0, max, c)$$

has the required properties. ∎

The next lemma shows that if we are given c_d then we an compute c_{d+1}.

Lemma 6.5 *Let* $\text{NEXT}(d, c, d', c')$ *mean that* $d' = d + 1$ *and if* $c = c_d$ *then* $c' = c_{d+1}$. *Then* NEXT *is expressible in (FO + pos TC).*

Proof We will express NEXT in (FO + pos TC), using PATH and NOPATH. In order to do this we must cycle through all vertices v, keeping a count of how many of them are reachable in at most $d + 1$ steps. We will say that there is a β edge from $\langle v, k\rangle$ to $\langle v', k'\rangle$ if and only if $v' = v + 1$ and $k' = k + 1$ if $\text{PATH}(v', d + 1)$ holds, and $k' = k$ otherwise. The formal definition of β is:

$$\begin{aligned}\beta(v, k, v', k'; d, c) \equiv\ & \Big(s(v, v') \wedge \\ & \Big[\Big(k = k' \wedge (\forall z)((z \neq v' \wedge \neg E(z, v')) \vee \text{NOPATH}(z, d, c)\Big) \\ & \vee \Big(s(k, k') \wedge (\exists z)((z = v' \vee E(z, v')) \wedge \text{PATH}(z, d)\Big)\Big]\Big)\end{aligned}$$

It follows that there is a β path from $\langle 0, 0\rangle$ to $\langle max, k\rangle$ iff $k = c_{d+1}$. Thus,

$$\text{NEXT}(d, c, d', c') \equiv s(d, d') \wedge (\text{TC}_{vk,v'k'}\ \beta)(0, 0, max, c')$$

∎

Let $\text{COUNT}(d, c)$ mean that $c = c_d$. This can be expressed as a transitive closure of NEXT:

$$\text{COUNT}(d, c) \equiv (\text{TC}_{dc,d'c'}\ \text{NEXT})(0, 0, d, c)$$

Finally, as promised we can express the nonexistence of a path from 0 to x:

$$(\exists c)(\text{COUNT}(max, c) \wedge \text{NOPATH}(x, max, c))$$

∎

One can generalize Theorem 6.3 to the following two corollaries. Corollary 6.7 settles a question dating back to 1964. These corollaries are surprising since almost everyone had conjectured their negation.[9]

Corollary 6.6 *[19, 30] For any $s(n) \geq \log n$,*

$$\textit{NSPACE}[s(n)] = \textit{co-NSPACE}[s(n)] .$$

Corollary 6.7 *[19, 30] The class of context sensitive languages is closed under complementation.*

Proof Kuroda showed in 1964 that CSL = NSPACE[n] [22]. ∎

References

[1] D. Mix Barrington, N. Immerman, and H. Straubing, *On Uniformity Within* NC^1, Third Annual Structure in Complexity Theory Symp. (1988).

[2] A. Borodin. S.A. Cook, P.W. Dymond, W.L. Ruzzo, and M. Tompa, *Two Applications of Complementation via Inductive Counting,* Third Annual Structure in Complexity Theory Symp. (1988).

[3] Ashok Chandra and David Harel, *Structure and Complexity of Relational Queries,* 21st IEEE Symp. on Foundations of Computer Science, (1980), (333-347). Also appeared in a revised form in JCSS 25 (1982), (99-128).

[4] Ashok Chandra, Dexter Kozen, and Larry Stockmeyer, *Alternation,* JACM, **28**, No. 1, (1981), 114-133.

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

[9] These two corollaries have been discovered independently and essentially simultaneously by Róbert Szelepcsényi [30].

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

[7] H. Enderton, *A Mathematical Introduction to Logic,* Academic Press, 1972.

[8] 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).

[9] Etienne Grandjean, *The Spectra of First-Order Sentences and Computational Complexity,* SIAM J. of Comp. **13**, No. 2 (1984), 356-373.

[10] Etienne Grandjean, *Universal quantifiers and time complexity of Random Access Machines,* Math. Syst. Th. (1985), 171-187.

[11] Etienne Grandjean, *First-order spectra with one variable,* to appear in J. Comput. Syst. Sci.

[12] Yuri Gurevich, *Toward Logic Tailored for Computational Complexity,* Computation and Proof Theory (M.M. Richter et. al., eds.). Springer-Verlag Lecture Notes in Math. 1104 (1984), 175-216.

[13] Neil Immerman, *Number of Quantifiers is Better than Number of Tape Cells,* JCSS **22**, No. 3, June 1981, 65-72.

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

[15] Neil Immerman, *Relational Queries Computable in Polynomial Time,* 14th ACM STOC Symp., (1982), 147-152. Also appeared in revised form in Information and Control, 68 (1986), 86-104.

[16] Neil Immerman, *Languages Which Capture Complexity Classes,* SIAM J. Comput. **16**, No. 4 (1987).

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

[18] Neil Immerman, *Expressibility and Parallel Complexity,* Tech. Report, Yale University Department of Computer Science (1987).

[19] Neil Immerman, *Nondeterministic Space is Closed Under Complementation,* to appear in the Third Structure in Complexity Theory Conf. (1988) and also in SIAM J. Comput.

[20] Neil Immerman and Dexter Kozen, *Definablitity with Bounded Number of Bound Variables,* Second LICS Symp. (1987)

[21] Neil Immerman and Eric S. Lander, "Describing Graphs: A First-Order Approach to Graph Canonization," Tech Report 605, Yale University Department of Computer Science (1988).

[22] S.Y. Kuroda, *Classes of Languages and Linear-Bounded Automata,* Information and Control 7 (1964), 207-233.

[23] K.J. Lange, B. Jenner, and B. Kirsig (1987), *The Logarithmic Hierarchy Collapses:* $A\Sigma_2^L = A\Pi_2^L$, 14th ICALP.

[24] Daniel Leivant, *Characterization of Complexity Classes in Higher-Order Logic,* Second Structure in Complexity Theory Conf. (1987), 203–217.

[25] James Lynch, "Complexity Classes and Theories of Finite Models," Math. Sys. Theory 15 (1982), 127-144.

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

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

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

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

[30] Róbert Szelepcsényi, *The Method of Forcing for Nondeterministic Automata,* Bull. European Association Theor. Comp. Sci. (Oct. 1987), 96-100.

[31] M. Vardi, *Complexity of Relational Query Languages,* 14th Symposium on Theory of Computation, 1982, (137-146).