Time, Hardware, and Uniformity

David Mix Barrington
Neil Immerman

ABSTRACT We describe three orthogonal complexity measures: parallel time, amount of hardware, and degree of non-uniformity, which together parametrize most complexity classes. We show that the descriptive complexity framework neatly captures these measures using three parameters: quantifier depth, number of variables, and type of numeric predicates, respectively. A fairly simple picture arises in which the basic questions in complexity theory — solved and unsolved — can be understood as questions about tradeoffs among these three dimensions.

1 Introduction

An initial presentation of complexity theory usually makes the implicit assumption that problems, and hence complexity classes, are linearly ordered by "difficulty". In the Chomsky Hierarchy each new type of automaton can decide more languages, and the Time Hierarchy Theorem tells us that adding more time allows a Turing machine to decide more languages. Indeed the word "complexity" is often used (e.g., in the study of algorithms) to mean "worst-case Turing machine running time", under which problems *are* linearly ordered.

Those of us who study structural complexity know that the situation is actually more complicated. For one thing, if we want to model parallel computation we need to distinguish between algorithms which take the same amount of "work" (i.e., sequential time) — we care how many processes are operating in parallel and how much parallel time is taken. These two dimensions of complexity are identifiable in all the usual models: boolean circuits (width and depth), PRAMs or other explicit parallel machines (number of processors and parallel time), alternating Turing machines (space and alternations) or even deterministic Turing machines (space and reversals). Hong's book [H] gives an interesting general treatment of "similarity" between models and "duality" between these two complexity measures.

Figure 1 gives a two-dimensional layout of some well-known (and less well-known) complexity classes. To be precise about our axes, we have chosen (unbounded fan-in) circuit depth as our measure of "parallel time" and circuit *width* as our measure of "number of processes". This assumes that the circuits have been arranged into levels, and that each edge into

$2^{n^{O(1)}}$	PH	?	?	PSPACE	?	EXPTIME
$2^{\log n^{O(1)}}$	qAC^0	?	qNC	?	qP	?
$n^{O(1)}$	AC^0	AC^1	NC	P	?	PSPACE
$(\log n)^{O(1)}$				SC	qNC	PSPACE
$O(\log n)$				LOGSPACE	qNC	PSPACE
$O(1)$				NC^1	qNC	PSPACE
$\uparrow$	$O(1)$	$O(\log n)$	$(\log n)^{O(1)}$	$n^{O(1)}$	$2^{\log n^{O(1)}}$	$2^{n^{O(1)}}$
width	**d**	**e**	**p**	**t**	**h**	$\rightarrow$

Figure 1: Two Dimensions of Complexity Classes

a gate comes from either an input or a gate on the immediately previous level. Thus the depth is the number of levels and the width is defined as the size of the largest level.

The columns of our chart represent bounds on depth, and the rows bounds on width. Blanks indicate classes where both are bounded below polynomially, and thus the resulting circuits cannot access the entire input. The named classes are fairly standard, except for the use of a prefix "q" to indicate a change from a polynomial to a quasipolynomial size bound [B92]. Thus qNC is the class of languages decidable by circuit families of poly-log depth and quasipolynomial size — it is a robust class which occurs several times on the chart. Finally, question marks denote classes which have no distinctive names known to the authors, and about which we know nothing other than the obvious containment relations with their neighbors.

Of course merely specifying combinatorial bounds on the circuits in a family does not fully specify a complexity class. For example, any unary language, even an uncomputable one, has a circuit family of $O(1)$ size and depth which decides it. In the circuit context, we usually speak of restricting circuit families by a *uniformity condition* — we say that the circuit must be computable (or that questions about it must be answerable) by resource-bounded computation. It is equally sensible to speak of *non-uniformity* as a resource, more of which allows a circuit family to decide more languages. This resource forms the third axis of our parametrization of complexity classes. It exists in other models as well — "advice" given to Turing machines [KL], or precomputation in parallel machines [A].

At each point on our two-dimensional chart, we have a range of complexity classes obtained by varying the uniformity condition. For example, if both size and depth are polynomially bounded the chart indicates the class P of languages decided by polynomial-time Turing machines. This claim is true if the circuits are P-uniform (computable by a poly-time Turing machine), or if they are DLOGTIME uniform (direct connection language decidable by a random-access Turing machine in time $O(\log n)$, see [BIS]), or any uniformity condition in between. However, if we allow ourselves more than polynomial time to compute the circuit, we may be able to decide more languages. (If we allow ourselves enough extra time, we can definitely do so. For example, if we allow more than exponential time, we can decide the unary version of the universal language for Turing machines with some superpolynomial time bound). On the other hand, one can imagine uniformity conditions so restrictive that a general simulation of a Turing machine is impossible.

In general as we pass upward (adding more non-uniformity) along the third axis we pass through three regions: one with too little non-uniformity, where the basic constructions relating the circuit model to other models cannot be carried out, a second robust region where a wide range of definitions give the same class, and a third region where additional non-uniformity gives steadily larger classes. The distinction can be quite important, as we see in the case of the class NC^1. As shown in [BIS], we can define a very restrictive uniformity notion under which NC^1 becomes the class of regular languages. If our non-uniformity resource is between DLOGTIME and NC^1 itself, we get a robust class, equal to ALOGTIME. And if we allow polynomial time to build our circuits, we can then do integer division and related problems [BCH] which (as far as we know) we couldn't do before.

This may be an example of where non-uniformity can replace one of the other two resources. In some cases, we know of limits on the potential power of non-uniformity to do so, at least subject to complexity-theoretic assumptions. For example, Karp and Lipton [KL] have shown that no amount of non-uniformity can allow P to simulate all of (uniform) NP, unless the polynomial hierarchy collapses to the second level. It would be interesting to have a parallel result for P and NC, and recent work of Ogihara, Cai, and Sivakumar [O, CS] has made progress toward this. One would like to derive unlikely complexity-theoretic consequences from, for example, the hypothesis that non-uniform NC^1 contains uniform P, or equivalently that there is a sparse set complete for P under NC^1 *Turing* reductions. Cai and Sivakumar show that if there is a sparse set complete for P under (logspace uniform) NC^1 *many-one* reductions, then P is equal to (logspace uniform) NC^1. D. van Melkebeek [vM] has subsequently shown a similar result for sparse sets complete under *truth-table* reductions.

In general our techniques for proving lower bounds on circuit complexity are combinatorial and algebraic and apply to the *totally non-uniform* versions of the circuit classes. A notable exception is the result by Allender and Gore [AG] that the integer permanent function is not in DPOLYLOGTIME-uniform $q\mathrm{ACC}^0$, though for all we know it might be in LOGSPACE-uniform ACC^0.

There is a certain amount of oversimplification in thinking of each of our three parameters as a single axis. For one thing, our "non-uniformity" resource is defined in terms of the "complexity" of languages which talk about the circuits, so this dimension may be as non-linear as the whole picture. However, the uniformity conditions we normally consider happen to be linearly ordered. More importantly, the "parallel time" axis is defined in terms of particular primitive operations on the data. In the circuit model a single gate computes an AND or OR, in a parallel machine the most powerful operations are the concurrent read and concurrent write, and the alternations of a Turing machine are also defined in terms of AND and OR. But there is nothing sacred about AND and OR — in each model we can consider other operations, such as MAJORITY (more powerful than AND and OR) or modular counting (orthogonal to them). In the circuit model these operations are embodied as new kinds of gates, in Turing machines as new acceptance conditions (as in the classes $\oplus$P or PP) and in parallel machines as new global operations (such as the "scan" operation on the Connection Machine).

Here we consider these three dimensions and the variety of operations in the framework of *descriptive complexity*, where we measure the complexity of a language by the syntactic resources (in a particular logical formalism) needed to express the property of membership in it [I87, I89, I89b]. (We review this framework in Section 2 below.) It has been known for some time that two parameters in descriptive complexity, number of variables and quantifier depth, correspond exactly to space and parallel time in either the circuit or PRAM models [I89b]. More recently (along with Straubing) [BIS, B90], we have extended the framework to deal with the third dimension and with more general operations, in the context of first-order formulas (or constant-depth, poly-size circuits). There varying uniformity conditions correspond to new atomic predicates and any associative operation with an identity can be modeled by a new type of quantifier.

With this approach, one can deal with a uniform family of circuits in terms of a single logical formula which defines the entire family. It is then possible to speak of "logically uniform circuits", and the traditional uniformity conditions which happen to coincide with logical uniformity (such as DLOGTIME uniformity for constant-depth circuits) are thus better motivated. Furthermore, proofs using logical uniformity are arguably simpler — two examples of this are the uniformity [AG, B92] of the upper bounds on the power of ACC^0 [Y, BT, GKR] and the relationship between threshold circuits and algebraic circuits over $GF(2^n)$ [Re, BFS, FVB].

In this paper we show that the descriptive complexity framework can deal with all three dimensions and with general operations, in virtually any possible combination. Specifically:

- In Section 2 we review the descriptive complexity framework and define quantifiers for arbitrary operations.

- In Section 3 we prove that the relationship of [I89b] between descriptive complexity, circuits, and PRAMs holds in the presence of these arbitrary operations — adding the new quantifiers to the logical formalism corresponds exactly to adding a new type of gate to the circuits or a new global operation to the PRAMs. If both circuit depth and width are polynomially bounded, the circuits are DLOGTIME uniform.

- In Section 4 we show that adding limited second-order variables to the formalism corresponds to increasing the size bound on the boolean circuits (or the processor bound on the PRAMs). This extends the treatment of quasipolynomial-size circuit classes in [B92].
- In Section 5 we show that adding new atomic predicates to the formalism corresponds to allowing the boolean circuits to be less uniform. This is true both in the polynomial-size and in larger-size domains.
- In Section 6 we attempt to extend the logical framework to describe circuit widths of less than polynomial (or equivalently, variables totaling to $o(\log n)$ bits). The main idea is to allow only variables which range over subpolynomial-size sets, but there are complications because the basic logical language assumes, for example, that it is possible to refer to every position in the input, which normally entails a variable ranging from 1 to n. Though it is not entirely satisfactory, we do develop enough machinery to bring the constant-width characterizations of NC^1 [B89] and PSPACE [CF] into the framework.

2 Background: Descriptive Complexity

In this section we will give an overview of the basic definitions of descriptive complexity theory. For more detailed presentations of this same material, see [I87] or [I89]. We will closely follow the development in [IL], especially for the notions of reductions and operators.

Any complexity theory starts with a formally defined set of problems, a model of computation, and a set of resource bounds. Our central notion will be to replace *deciding* a problem by *describing* it. Our "model of computation" is to write down a logical formula which is true of an input exactly when the decision problem has a positive answer. Our resource measures will then be properties of the logical formula, such as the number and type of quantifiers, variables, and atomic predicates it contains.

Our general computational *problem* is to take some input and return a yes or a no answer. We need to have the input in a form which our formulas can talk about, so we code all our inputs as *finite logical structures.* For example, suppose the input is a binary string $w = w_1 \ldots w_n$ of length n. Our atomic statements about this string will be to name the bit in position i, which we will do by a predicate $M(i)$. Following the traditions of mathematical logic, we code the string as a *structure* $\mathcal{A}_w$ with *universe* $\{1, 2, \ldots, n\}$ (the input positions) and one unary *relation* M on this universe. Thus $\mathcal{A}_w = \langle \{1, 2, \ldots, n\}, M^{\mathcal{A}_w} \rangle$, where,

$$M^{\mathcal{A}_w} \quad = \quad \{ i \mid w_i = 1 \}$$

On the other hand, our problem might be a set of directed graphs rather than a set of strings. In this case our structures would have a universe consisting of the vertices and a binary relation E on this universe such that $E(i,j)$ is true if and only if there is an edge from i to j. Or our input might be an entire relational database, with several different relations defined on a fixed universe. What we need to know before we can describe a problem is exactly which relation symbols (and constant symbols, which may be thought of as 0-ary function symbols) are available in our language.

In general, a *vocabulary*

$$\tau = \langle R_1^{a_1}, \ldots, R_t^{a_t}, c_1, \ldots, c_s \rangle$$

is any tuple of input relation symbols and constant symbols. For example, the vocabulary of strings is $\tau_s = \langle M^1 \rangle$ consisting of a single monadic relation symbol and the vocabulary of graphs $\tau_g = \langle E^2, s, t \rangle$ consists of the binary edge predicate and two constant symbols.

To define a *structure* $\mathcal{A}$ over vocabulary τ, we need to provide a finite universe, a table for each relation $R_i^{\mathcal{A}}$ and a value for each constant $c_i^{\mathcal{A}}$. We will use the notation $|\mathcal{A}|$ to denote the universe of $\mathcal{A}$, and $\|\mathcal{A}\|$ its cardinality. In this paper, all our universes will be ordered and we will thus assume they they consist of the first n positive integers,

$$|\mathcal{A}| \quad = \quad \{1, 2, \ldots, n\}, \quad \text{where } n = \|\mathcal{A}\|$$

Formally, a structure $\mathcal{A}$ of vocabulary τ is a tuple

$$\mathcal{A} = \langle \{1, 2, \ldots, n\}, R_1^{\mathcal{A}}, \ldots, R_t^{\mathcal{A}}, c_1^{\mathcal{A}}, \ldots, c_s^{\mathcal{A}} \rangle$$

Here $R_i^{\mathcal{A}}$ is a subset of $|\mathcal{A}|^{a_i}$ for each $i \leq t$ and $c_j^{\mathcal{A}}$ is an element of $|\mathcal{A}|$ for each $j \leq s$.

For any τ, define STRUC$[\tau]$ to be the set of all finite structures of vocabulary τ. Define a complexity theoretic *problem* to be any subset of STRUC$[\tau]$ for some τ. For example, a problem over binary strings is a subset of STRUC$[\tau_s]$ and a graph problem is a subset of STRUC$[\tau_g]$.

Next we need to build up a system $\mathcal{L}(\tau)$ of logical formulas to talk about a problem with vocabulary τ. Along with the symbols of τ we will always have a fixed set of numeric relation symbols and constant symbols: $=, \leq, \mathrm{SUC}, \mathrm{BIT}, \mathit{min}, \mathit{max}$.[1] We then allow the boolean connectives $\wedge, \vee, \neg$, we introduce variables: $x, y, z, \ldots$ ranging over $|\mathcal{A}|$, and allow first-order quantification of these variables by the usual quantifiers $\forall$ and $\exists$.

[1] Here $\leq$ refers to the usual ordering on the universe, $\{1, 2, \ldots, n\}$. SUC is the successor relation and *min* and *max* refer to the first and last elements in the total ordering. BIT(i,j) holds iff the j^{th} bit in the binary expansion of i is a one. These relations are called numeric because, for example, "BIT(i,j)" and "$i \leq j$" describe the numeric values of i and j and do not refer to any input predicates.

This definition gives us a descriptive complexity class FO(τ), consisting of all problems (subsets of STRUC[τ]) which can be defined using the first-order language $\mathcal{L}(\tau)$. When the vocabulary is understood, we'll call this class just FO. It turns out that this class is a familiar one from circuit complexity and parallel complexity — it equals the log-time uniform version of the circuit class AC^0 and also the problems solvable in constant time by a natural parallel machine (a version of a PRAM) with polynomially many processors [BIS].

All of our later descriptive complexity classes may be viewed as augmentations of this class FO. As a first example, consider adding a new quantifier to $\exists$ and $\forall$. We define the majority quantifier M so that $(Mx)\varphi(x)$ holds if more than half the elements of the universe satisfy $\varphi(x)$. With this new tool in our first-order language, we can define a larger set of problems called FOM. This class is also familiar, as it is equal to the log-time uniform version of ThC^0, the problems solvable by constant-depth, polynomial-size threshold circuits [BIS]. After building up some more machinery, we will describe a general method to produce new operators of this kind.

First-Order Reductions

In this section we define first-order reductions, which provide the most natural way to reduce one problem to another in the descriptive setting. They are exactly many-one reductions which are definable by first-order formulas. At the end of the section we will also define some important subclasses of first-order reductions.

Suppose that $S \subseteq \mathrm{STRUC}[\sigma]$ and $T \subseteq \mathrm{STRUC}[\tau]$ are any two problems. Let $\tau = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$. In order to reduce S to T, we must provide a way for any structure $\mathcal{A}$ to be mapped to a new structure $I(\mathcal{A}) \in \mathrm{STRUC}[\tau]$. Before giving the definition, we provide an example.

Example 2.1 Let graph reachability (REACH) denote the following problem: given a graph, G, and vertices s, t, determine if there is a path from s to t in G. Let REACH_u be the restriction of the REACH problem to undirected graphs. Let REACH_d be the restriction of the REACH problem in which we only allow deterministic paths, i.e., if the edge (u, v) is on the path, then this must be the unique edge leaving u. Notice that the REACH_d problem is reducible to the REACH_u problem as follows: Given a directed graph, G, let G' be the undirected graph that results from G by the following steps:

1. Remove all edges out of t.

2. Remove all multiple edges leaving any vertex.

3. Make each remaining edge undirected.

Observe that there is a deterministic path in G from s to t iff there is a path from s to t in G'.

The following first-order formula φ_{du} accomplishes these three steps and is thus a first-order reduction from REACH_d to REACH_u. More precisely, the first-order reduction is the expression $I_{du} = \lambda_{xy}(\mathbf{true}, \varphi_{du}, s, t)$ whose meaning is, "Make the new edge relation $\{(x,y) \mid \varphi_{du}\}$, and map s to s and t to t."

$$\begin{aligned} \alpha(x,y) &\equiv E(x,y) \wedge x \neq t \wedge (\forall z)(E(x,z) \rightarrow z = y) \\ \varphi_{du}(x,y) &\equiv \alpha(x,y) \vee \alpha(y,x) \end{aligned}$$

The reason for the formula "**true**" is that we want to put all elements of the input structure into the output structure,

$$|I(\mathcal{A})| \quad = \quad \big\{g \in |\mathcal{A}| \bigm| \mathcal{A} \models \mathbf{true}\big\} \quad = \quad |\mathcal{A}|$$

□

In this example the universes of the domain and range of the reduction were equal, and hence had the same size. The general situation is a bit more complicated because we want to allow the possibility of reducing one problem to another which has a polynomially larger universe. Here we fix an integer k and let $|I(\mathcal{A})|$ be a first-order definable set of k-tuples of elements of $|\mathcal{A}|$. Each relation $R_i^{a_i}$ of τ has a formula φ_i whose free variables are $x_1^1, \ldots, x_1^k, \ldots, x_{a_i}^1, \ldots, x_{a_i}^k$ (which may be viewed as a_i free variables each of which is a k-tuple). Similarly, each constant symbol c_j of τ has associated with it a k-tuple $t_j^1, \ldots, t_j^k$ of constants from σ. We recapitulate by giving the formal definition:

Definition 2.2 (First-Order Reductions) Let σ and τ be be two vocabularies, with $\tau = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s\rangle$. Let $S \subseteq \mathrm{STRUC}[\sigma]$ and $T \subseteq \mathrm{STRUC}[\tau]$ be two problems. Let k be a positive integer. Let

$$I = \lambda_{x_1^1, \ldots, x_d^k}\langle \varphi_0, \varphi_1, \ldots, \varphi_r, \overline{t_1}, \ldots, \overline{t_s}\rangle$$

be a tuple consisting of an $r+1$-tuple of formulas and an s-tuple of k-tuples of constants, all from $\mathcal{L}(\sigma)$. Here $d = \max_i(a_i)$.

Then I induces a mapping I from $\mathrm{STRUC}[\sigma]$ to $\mathrm{STRUC}[\tau]$ as follows. Let $\mathcal{A} \in \mathrm{STRUC}[\sigma]$ be any structure of vocabulary σ, and let $n = \|\mathcal{A}\|$. Then the structure $I(\mathcal{A})$ is defined as follows:

$$I(\mathcal{A}) \ = \ \langle |I(\mathcal{A})|, R_1^{I(\mathcal{A})}, \ldots, R_r^{I(\mathcal{A})}, c_1^{I(\mathcal{A})}, \ldots, c_s^{I(\mathcal{A})}\rangle$$

The universe is given by,

$$|I(\mathcal{A})| \quad = \quad \big\{(g_1, \ldots, g_k) \in |\mathcal{A}|^k \bigm| \mathcal{A} \models \varphi_0(g_1, \ldots, g_k)\big\}$$

That is, the universe is the set of k-tuples of $\mathcal{A}$ satisfying φ_0. The ordering of $|I(\mathcal{A})|$ is the lexicographic ordering inherited from $\mathcal{A}$. Each $c_j^{I(\mathcal{A})}$ is given by the k-tuple of constants $(t_j^1, \dots, t_j^k)$. The relation $R_i^{I(\mathcal{A})}$ is determined by the formula φ_i, for $i = 1, \dots, r$:

$$R_i^{I(\mathcal{A})} \;=\; \{((u_1^1, \dots, u_1^k), \dots, (u_{a_i}^1, \dots, u_{a_i}^k)) \;|\; \mathcal{A} \models \varphi_i(u_1^1, \dots u_{a_i}^k)\}$$

If the structure $\mathcal{A}$ interprets some variables $\bar{u}$ then these may appear freely in the the φ_i's and t_j's of I, and the definition of $I(\mathcal{A})$ still makes sense. This will be important in Definition 2.5 where we define operators in terms of first-order reductions.

Suppose that I is a many-one reduction from S to T, i.e. for all $\mathcal{A}$ in STRUC$[\sigma]$,

$$\mathcal{A} \in S \quad \Leftrightarrow \quad I(\mathcal{A}) \in T$$

Then we say that I is a k-ary *first-order reduction* of S to T. Furthermore, if the φ_i's are quantifier-free and do not include BIT then I is a *quantifier-free reduction.* □

Valiant [V] defined a very low-level, non-uniform reduction called a *projection.* A projection is a many-one reduction $f : \{0,1\}^\star \to \{0,1\}^\star$ such that each bit of $f(w)$ depends on at most one bit of w. (It can be thought of as a reduction computed by a circuit of depth *zero*, depending on the details of the definitions.) We next define first-order projections, a syntactic restriction of first-order interpretations.

Definition 2.3 (First-Order Projections) Let I be a k-ary first-order reduction from S to T as in Definition 2.2. Let $I = \langle \varphi_0, \dots, \varphi_r, \overline{t_1}, \dots, \overline{t_s} \rangle$. Suppose further that the φ_i's all satisfy the following *projection condition:*

$$\varphi_i \equiv \alpha_1 \vee (\alpha_2 \wedge \lambda_2) \vee \cdots \vee (\alpha_s \wedge \lambda_s) \qquad (2.4)$$

where the α_j's are mutually exclusive formulas in which no input relations occur, and each λ_j is a literal, i.e. an atomic formula $P(x_{j_1}, \dots x_{j_a})$ or its negation.

In this case the predicate $R_i(\langle u_1^1, \dots, u_1^k \rangle, \dots, \langle u_{a_i}^1 \dots, u_{a_i}^k \rangle)$ holds in $I(\mathcal{A})$ if $\alpha_1(\bar{u})$ is true, or if $\alpha_j(\bar{u})$ is true for some $1 < j \le t$ and the corresponding literal $\lambda_j(\bar{u})$ holds in $\mathcal{A}$. Thus each bit in the binary representation of $I(\mathcal{A})$ is determined by at most one bit in the binary representation of $\mathcal{A}$. We say that I is a *first-order projection.*

Finally define a *quantifier-free projection* to be a first-order projection that is also a quantifier-free reduction. Write $S \le_{\text{fop}} T$, $S \le_{\text{qfp}} T$ to mean that S is reducible to T via a first-order projection, respectively a quantifier-free projection. □

Looking back at Example 2.1, we see that I_{du} is a quantifier-free reduction, but it is not a projection. This is because the formula $\varphi_{du}(x, y)$ looks at more than one bit of its input: it depends on $E(x,y)$ and $E(y,x)$ and in fact it may depend on all $E(x,z)$ and $E(y,z)$ as z varies over all vertices. In fact, there is a qfp that reduces REACH_d to REACH_u, but we will not construct it here.

General Operators

First-order reductions give us a mechanism for forming an operator out of *any* problem Θ. The operator, which we will also call Θ, acts rather like a quantifier (and in fact this construction extends the "generalized quantifiers" of [BIS]). A very similar construction is used in [MP] and in [KV], where the operator we call Θ would be denoted Q_Θ.

Definition 2.5 ([IL]) (Operator Form of a Problem) Let σ and τ be vocabularies, and let $\Theta \subseteq \text{STRUC}[\tau]$ be any problem. Let I be any first-order reduction with $I : \text{STRUC}[\sigma] \to \text{STRUC}[\tau]$. Then $\Theta[I]$ is a well-formed formula in the language $\text{FO}(\Theta)$ over vocabulary σ, with the semantics:

$$\mathcal{A} \models \Theta[I] \quad \Leftrightarrow \quad I(\mathcal{A}) \in \Theta \qquad \square$$

For example, consider the case where $\tau = \tau_s$ and so Θ is set of binary strings. A k-ary reduction I then consists of a pair of formulas $\langle \varphi_0, \varphi_1 \rangle$ from $\mathcal{L}(\sigma)$, with free variables $x^1, \dots x^k$. $\Theta[I]$ is a sentence that holds of a structure $\mathcal{A} \in \text{STRUC}[\sigma]$ exactly if the structure $I(\mathcal{A})$ is in Θ. This structure $I(\mathcal{A})$ is the string of length at most $|\mathcal{A}|^k$ consisting of the truth values of φ as the variables $x^1, \dots x^k$ range over $\left\{\bar{a} \in |\mathcal{A}|^k \;\middle|\; \mathcal{A} \models \varphi_0(\bar{a})\right\}$. The operator Θ is a generalized quantifier binding the variables $x^1, \dots x^k$.

We have already seen three special cases of this phenomenon — choosing Θ to be the AND, OR, or MAJORITY languages gives the $\forall$, $\exists$, and M quantifiers respectively. The generalized quantifiers in [BIS], corresponding to languages over larger alphabets, fit into this framework with τ having a unary predicate for each letter of the language.

We can repeat this process, applying the Θ operator to a formula in which Θ already appears. We define the complexity class $\text{FO}(\Theta)$ to be the set of problems expressible by first-order formulas in this extended language. We say that a problem S is *first-order Turing reducible* to Θ ($S \le^t_{\text{fo}} \Theta$) iff $S \in \text{FO}(\Theta)$.

If the problem Θ is complete for $\mathcal{C} = \text{FO}(\Theta)$ via a lower-level reduction such as fop or qfp this means that we do not need the full power of arbitrary applications of Θ in $\text{FO}(\Theta)$. In other words, a normal form theorem for $\text{FO}(\Theta)$ applies:

Definition 2.6 ([IL]) We say that the language $\text{FO}(\Theta)$ has the *fop Normal Form Property* iff every formula $\varphi \in \text{FO}(\Theta)$ is equivalent to a formula ψ, where,

$$\psi \quad = \quad \Theta[I]$$

where I is a first-order projection. If I is a quantifier-free projection then we say that FO(Θ) has the *qfp Normal Form Property.* □

As an example, recall the problem REACH from Example 2.1. Consider the following quantifier-free projection I_c:

$$I_c = \lambda_{x^1,x^2,y^1,y^2}(\mathbf{true}, \alpha, \langle min, min\rangle, \langle max, max\rangle)$$

$$\alpha \equiv (x^1 = y^1 \wedge E(x^2,y^2)) \vee (x^1 = x^2 \wedge \mathrm{SUC}(x^1,y^1) \wedge y^2 = min)$$

The formula α describes a graph on pairs of vertices. There is an α-path from $\langle min, min\rangle$ to $\langle max, max\rangle$ iff every vertex in the graph is reachable from *min.* For undirected graphs, this is equivalent to connectivity,

$$\text{CONNECTIVITY} \quad \equiv \quad \text{REACH}(I_c) \tag{2.7}$$

It is shown in [I87] and [I88] that the language FO(TC) is equal to NL and has the qfp normal form property. It thus follows that every problem in NL is expressible in the form of Equation 2.7, with I_c replaced by other qfp's.

The role of arbitrary problems as generalized quantifiers is now summarized by:

Fact 2.8 ([IL]) *Let Θ be a problem and $\mathcal{C}$ a complexity class that is closed under first-order Turing reductions. Then*

1. *Θ is $\le^t_{\mathrm{fo}}$-complete for $\mathcal{C}$ if and only if $\mathcal{C} = \mathrm{FO}(\Theta)$.*
2. *Θ is $\le_{\mathrm{fop}}$-complete for $\mathcal{C}$ if and only if $\mathcal{C} = \mathrm{FO}(\Theta)$ and $\mathrm{FO}(\Theta)$ has the fop normal form property.*
3. *Θ is $\le_{\mathrm{qfp}}$-complete for $\mathcal{C}$ if and only if $\mathcal{C} = \mathrm{FO}(\Theta)$ and $\mathrm{FO}(\Theta)$ has the qfp normal form property.*

3 First Uniformity Theorem

An apparent limitation of first-order logic as a means of describing problems is that a single formula has only a fixed number of quantifiers and can thus *a priori* only represent a language which is decided by circuits of constant depth. Adding operators for new functions expands the expressible problems considerably, but in order to deal with circuits with arbitrary depth bounds we need the notion of *iterated quantifier blocks.* Rather than a single formula, we now have a family of formulas varying with the input size n, but in a very predictable way. For any function $t(n)$, we let our formula consist of $t(n)$ syntactic copies of a block of quantifiers, followed by some base formula. It is important to note that when we repeat a quantifier block the variables are not changed. Thus an FO$[t(n)]$ formula uses only a bounded number of distinct variables. For more detail see [I89].

Definition 3.1 ([I89]) A set C of structures of vocabulary τ is a member of $FO[t(n)]$ iff there exist quantifier-free formulas M_i, $0 \le i \le k$, from $\mathcal{L}(\tau)$, and a quantifier block,

$$\mathrm{QB} \;=\; \big[(Q_1 v_1 . M_1) \ldots (Q_k v_k . M_k)\big]$$

such that if we let $\varphi_n = [\mathrm{QB}]^{t(n)} M_0$, for $n = 1, 2, \ldots$, then for all structures $\mathcal{A}$ of vocabulary τ with $\|\mathcal{A}\| = n$, $\quad \mathcal{A} \in C \quad \Leftrightarrow \quad \mathcal{A} \models \varphi_n$. □

This covers only ordinary quantifiers — we now generalize Definition 3.1 to get the class FO$(\Theta)[t(n)]$, by allowing some of the quantifiers Q_i to be applications of Θ.

Recall that the operator Θ should be used in the form $\Theta[I]$, where I is a first-order reduction (Definition 2.2),

$$I = \lambda_{x_1 \ldots x_d} \langle \varphi_0, \varphi_1, \ldots, \varphi_r, t_1, \ldots, t_s \rangle$$

We will write such an occurrence of Θ in the quantifier block as

$$(\Theta b^r, x_1 \ldots x_d; t_1, \ldots, t_s),$$

where we use a tuple of boolean variables b^r to code a value between 0 and r. This expression thus binds the boolean variables b^r, and the individual variables $x_1 \ldots x_d$. The meaning is given by:

$$(\Theta\, b^r, x_1 \ldots x_d; t_1 \ldots t_s)(\gamma) \quad \equiv \Theta(\langle \gamma(b^r/0), \gamma(b^r/1), \ldots, \gamma(b^r/r), t_1, \ldots, t_s \rangle)$$

where $\gamma(b^r/i)$ denotes the substitution of the number i for the boolean variables b^r in γ.

Definition 3.2 (FO$(\Theta)[t(n)]$) A set C of structures of vocabulary τ is a member of FO$(\Theta)[t(n)]$ iff there exist generalized quantifiers $G_1, G_2, \ldots, G_k$ and quantifier free formula M_0 from $\mathcal{L}(\tau)$, and a quantifier block,

$$\mathrm{QB} \;=\; \big[(G_1) \ldots (G_k)\big]$$

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

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

Here, each generalized quantifier G_i is either a limited quantifier:

$$G_i \quad = \quad (Q_i v_i . M_i)$$

as in Definition 3.1; or an application of Θ:

$$G_i \quad = \quad (\Theta\, b^r, x_1 \ldots x_d; t_1 \ldots t_s)$$ □

Similarly, we define $\mathrm{IND}(\Theta)[t(n)]$ to be the set of problems definable by first-order inductive definitions with Θ operators, with inductive depth at most $t(n)$. Extending the argument in [I89], this complexity class is equivalent to the restriction of $\mathrm{FO}(\mathrm{LFP}, \Theta)$ in which all applications of LFP have depth of nesting at most $t(n)$. A typical example, as we will see, is that the language $\mathrm{FO}(M)[\log n]$ consisting of quantifier-blocks including the majority quantifier, iterated $\log n$ times, is equal to the class ThC^1 of problems accepted by threshold circuits of depth $O(\log n)$.

Our main result in this section, generalizing basic results in [BIS] and [IL], is that the known relationships between first-order descriptive complexity classes and other standard parallel complexity classes are not affected by the introduction of the Θ operators. This result will be the basis of all our discussions of uniformity in the remainder of the paper. First, however, we must specify the exact definitions of our various parallel models.

Following [I89b], we choose as our parallel machine model the CRCW-PRAM with polynomially much hardware. The complexity class $\mathrm{CRAM}[t(n)]$ is the set of problems solvable by such a CRCW-PRAM in parallel time $t(n)$. To generalize, we may allow the CRAM to have special hardware that can execute the operator Θ in constant time. We call the problems solvable in $t(n)$ parallel time by this augmented machine the class $\mathrm{CRAM}(\Theta)[t(n)]$.

The circuit complexity class $\mathrm{AC}(\Theta)[t(n)]$ is the set of problems solvable by depth-$t(n)$ circuits with AND, OR, and Θ gates of unbounded fan-in. Note that since Θ need not be a symmetric function, the string or structure defining the circuit must somehow specify the order of input to the Θ gates. The uniformity condition, then, will constrain the difficulty of computing the predicate $\mathrm{IN}\#(g, h, i)$, meaning that gate h is input number i of gate g. (This predicate is the natural generalization of the "direct connection language" of [Ru, BIS].)

A circuit is a directed, acyclic graph. The leaves of the circuit are the input nodes. Every other vertex is a gate. The edges of the circuit indicate connections between nodes. The edge (a, b) would indicate that the output of gate a is an input to gate b.

Define the vocabulary of circuits, $\tau_c = \langle E^2, \mathrm{IN}\#^3, L^1, G^1_\wedge, G^1_\vee, G^1_\neg, G^1_t, r\rangle$, where $E(x, y)$ is a directed edge relation, meaning that the output of node x is an input to node y, $L(x)$ means that the leaf x takes input value "1", $G_\wedge(x)$, $G_\vee(y)$, $G_\neg(z)$, and $G_t(w)$ mean that the nodes x, y, z, and w are "and", "or", "not", and "Θ" gates, respectively. The constant r refers to the root node, or output of the circuit.

In (3) below "first-order uniform" means there is a first-order reduction, $I : \mathrm{STRUC}[\tau_s] \to \mathrm{STRUC}[\tau_c]$. The n^{th} circuit is given by $\mathcal{C}_n = I(0^n)$, where $0^n \in \mathrm{STRUC}[\sigma_s]$ is the string consisting of n zeros. In (4), the predicates of $\mathcal{C}_n$ must all be computable in time $DTIME(\log n)$, where $n = \|\mathcal{A}\|$ is the size of the input.

In [SV], the non-uniform versions of $\mathrm{CRAM}[t(n)]$ and $\mathrm{AC}[t(n)]$ were shown to be equal. Since then, many other connections have been established and they remain true in the uniform setting.

Theorem 3.3 *For constructible and polynomially bounded $t(n)$, and any problem operator Θ, the following classes are equal:*

1. $\mathrm{FO}(\Theta)[t(n)]$
2. $\mathrm{IND}(\Theta)[t(n)]$
3. *First-order uniform* $\mathrm{AC}(\Theta)[t(n)]$
4. *DLOGTIME uniform* $\mathrm{AC}(\Theta)[t(n)]$
5. $\mathrm{CRAM}(\Theta)[t(n)]$

Proof

$(1 = 2 = 5)$: This is very similar to the proof of the same fact without Θ [I89b]. The proof that $\mathrm{IND}(\Theta)[t(n)]$ contains $\mathrm{CRAM}(\Theta)[t(n)]$ is least different: we can inductively describe the entire configuration of the CRAM at some time $t+1$ in terms of its configuration at time t. In addition to all the other cases, when the CRAM uses the operation Θ, this is simulated by the inductive definition using the same operation. Similarly, that $\mathrm{CRAM}(\Theta)[t(n)]$ contains $\mathrm{FO}(\Theta)[t(n)]$ requires the same change: as the CRAM simulates the formula, the new operator Θ can be performed by the CRAM when it is invoked in the formula. Finally we show, by induction on the structure of the inductive definition, that $\mathrm{FO}(\Theta)[t(n)]$ contains $\mathrm{IND}(\Theta)[t(n)]$. The idea is that we can rewrite any positive first-order inductive definition into a quantifier block as in Definition 3.2. Again, the proof from [I89b] goes through with the change that occurrences of Θ in the inductive definition are copied into the quantifier block.

$(4 \subseteq 3)$: This is immediate since DLOGTIME is contained in FO [BIS].

$(3 \subseteq 2)$: Here we are given the first-order uniform $\mathrm{AC}(\Theta)[t(n)]$ circuits. We produce an $\mathrm{FO}(\Theta)$ inductive definition of the value of each gate in the circuit. The depth of the induction will be equal to the depth of the circuit.

We will define by induction in $\mathrm{FO}(\Theta)$ the predicate $\mathrm{VALUE}(g, b)$ whose intuitive meaning is that the value of the gate g is the boolean b. The inductive definition for VALUE will be a disjunction over the possible kinds of gates: leaf, $\wedge, \vee, \Theta$. Thus, the definition of VALUE is given as follows:

$$\begin{aligned}\mathrm{VALUE}(g,b) \quad \equiv \quad & \mathrm{DEFINED}(g) \ \wedge \ \big[G_L(g) \wedge (L(g) \leftrightarrow b) \ \vee \\ & \quad G_\wedge(g) \wedge (C(g) \leftrightarrow b) \quad \vee \quad G_\vee(g) \wedge (D(g) \leftrightarrow b) \ \vee \\ & \quad G_\neg(g) \wedge (N(g) \leftrightarrow b) \quad \vee \quad G_t(g) \wedge (T(g) \leftrightarrow b)\big]\end{aligned}$$

Here, $G_L(g)$, meaning that g is a leaf, is an abbreviation for $(\forall x)\neg E(x,g)$. $\mathrm{DEFINED}(g)$, meaning that g is ready to be defined, is an abbreviation for $(\forall x)(\exists c)(E(x,g) \rightarrow \mathrm{VALUE}(x,c))$.

The predicate $C(g)$ says that all of g's inputs are true, $D(g)$ says that some of g's inputs are true, and $N(g)$ says that its input is false:

$$\begin{array}{rcl} C(g) & \equiv & (\forall h)(E(h,g) \rightarrow \mathrm{VALUE}(h,1)) \\ D(g) & \equiv & (\exists h)(E(h,g) \wedge \mathrm{VALUE}(h,1)) \\ N(g) & \equiv & (\exists! h)(E(h,g)) \;\wedge\; (\exists h)(E(h,g) \wedge \mathrm{VALUE}(h,0)) \end{array}$$

The most interesting case is $T(g,b)$. In this case, the inputs to g code a set of relations that are a valid input to the problem Θ. Consider the simplest case, in which this problem is coded as a single relation, R. In this case, we have

$$\begin{array}{rcl} T(g) & \equiv & \Theta[I], \qquad \text{where, } I = \lambda_x \langle \varphi_0, \varphi_1 \rangle \\ \varphi_0(x) & \equiv & (\exists h)(\mathrm{IN}\#(g,h,x)) \\ \varphi_1(x) & \equiv & (\exists h)(\mathrm{IN}\#(g,h,x) \;\wedge\; \mathrm{VALUE}(h,1)) \end{array}$$

Note that the existence of the predicate IN# giving the numbering of the inputs to each gate is crucial here. In [BIS] the weaker condition was used that the ordering of the inputs was given. Then the assumption that all operators were "monoidal" was used: the appropriate formula was constructed by entering benign identity elements between the valid inputs. Since we are dealing with arbitrary operators Θ we have no choice but to insist on the existence of the numbering. Note also that if FO(Θ) contains the class ThC^0 then we can express the numbering of the inputs if we are given the ordering of the inputs.

($1 \subseteq 4$): Here we are given the quantifier-block which when iterated $t(n)$ times expresses the problem in question for structures of size n. We show how, in DLOGTIME, to recognize the direct connection language of the equivalent AC(Θ) circuits. The idea is that each gate in the circuit corresponds to a quantifier or boolean connective, or occurrence of Θ in the quantifier block, indexed by the values of all the (bounded number of) variables, and the time. All that we need from DLOGTIME is the power to compute the successor of a $\log n$-bit number, and to find a particular number in a bounded size tuple of $\log n$-bit numbers. □

There were two reasons for the requirement that $t(n)$ be polynomially bounded in Theorem 3.3. The first is that the definition of IND requires monotone inductive definitions, which automatically close in at most polynomially many steps. This can be alleviated by changing to an iteration operator (ITER) which does not require monotonicity. (This is equivalent to replacing the least fixed point operator (LFP) by a partial fixed point operator (PFP)). The second reason concerns the class DLOGTIME uniform AC(Θ)[$t(n)$]. For times $t(n)$ greater than polynomial, we need more than DTIME[$\log n$] exactly to read and copy variables of length more than $\log n$. Thus, to talk about uniformity for superpolynomial-size circuits, we interpret DLOGTIME uniform to mean DTIME[$\log(n + t(n))$]. Similarly, first-order uniform means that the circuits are first-order recognizable. This automatically implies that the first-order variables needed to describe such a circuit would be of size ($\log(n + t(n))$) bits. With these points modified as above, Theorem 3.3 remains true when the restriction that $t(n)$ be polynomially bounded is removed.

4 Variables That Are Longer Than $\log n$ Bits

We know from [I89b, I91] that the number v of $\log n$-bit variables in a formula corresponds approximately to the amount of hardware n^v in a circuit or CRAM. Thus, a constant number of $\log n$ bit variables gives us the usual bound of polynomial hardware. A constant number of *second-order* variables (i.e, variables with polynomially many bits) gives exponential hardware. In the following result, the correspondence is not perfect simply because the CRAM model, with priority write, is a different concurrent write model of parallelism from the FO[$t(n)$] model. Interestingly, in Fact 4.3, using the more robust measure of DSPACE, the bound is tight.

Fact 4.1 ([I89b]) *Let* CRAM[$t(n)$]$\cdot$PROC[$p(n)$] *be the complexity class* CRAM[$t(n)$] *restricted to machines using at most $O[p(n)]$ processors. Let* IND[$t(n)$]$\cdot$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 both $t(n)$ and the maximum size of a register word are $o[\sqrt{n}]$, and that $\pi \geq 1$ is a natural number. Then,*

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

Fact 4.2 ([I89b]) *The polynomial hierarchy* PH *is equal to the set of properties checkable by a* CRAM *using exponentially many processors and constant time:*

$$\mathrm{PH} \;=\; \bigcup_{k=1}^{\infty} \mathrm{CRAM}[1]\cdot\mathrm{PROC}[2^{n^k}]\,.$$

Fact 4.3 ([I91]) *For $k = 1, 2, \ldots,$*

$$\mathrm{DSPACE}[n^k] \;=\; \mathrm{VAR}[k+1]$$

In this section, we show how to define the number of variables in the FO[$t(n)$] model in such a way that even when the number of variables is more than a constant, the simple, uniform quantifier block structure is preserved. Then we use this new definition to generalize Theorem 3.3 to two of our three complexity "dimensions".

Definition 4.4 For $v(n) \geq 1$, a $\mathrm{FO}[t(n)] \cdot \mathrm{VAR}[v(n)]$ formula will have two sorts of variables: the domain variables: $x, y, z, \ldots$, ranging over the universe $\{1, 2, \ldots, n\}$, plus extended variables: $X, Y, Z, \ldots$, each of $v(n) \log n$ bits.

The extended variables may be quantified just like domain variables. However, the extended variables do not occur as arguments to any input relations. Their only role is as arguments in the BIT predicate. That is, we may assert $\mathrm{BIT}(x, Y)$ meaning that the x^{th} bit of Y is a one. Note that this only makes sense for extended variables with at most n bits. (In this paper we only consider polynomially bounded $v(n)$ for which we can use a tuple $\bar{x}$ of domain variables. One could consider even larger $v(n)$, by using intermediate size variables between x and Y.) Define $\mathrm{FO}[t(n)] \cdot \mathrm{VAR}[v(n)]$, with $v(n) \geq 1$ to be the extension to $\mathrm{FO}[t(n)]$ that we get by including a bounded number of $v(n) \log n$-bit extended variables.

Furthermore, extended variables can be used in the natural way to define reductions that increase the size of a structure by more than a polynomial. Thus, we have a definition of $\mathrm{FO}(\Theta)[t(n)] \cdot \mathrm{VAR}[v(n)]$ for any $v(n) \geq 1$. □

As an example illustrating the above definition, the next proposition says that second-order logic is first-order logic with polynomially many variables:

Proposition 4.5 *For any problem Θ and any function $t(n)$,*

$$\mathrm{SO}(\Theta)[t(n)] \; = \; \mathrm{FO}(\Theta)[t(n)] \cdot \mathrm{VAR}[n^{O(1)}]$$

We can now state our Two-Dimensional Resources Theorem:

Theorem 4.6 *For constructible $v(n) \geq 1$, constructible $t(n)$, and any problem operator Θ, the following classes are equal:*

1. $\mathrm{FO}(\Theta)[t(n)] \cdot \mathrm{VAR}[O(v(n))]$
2. $\mathrm{ITER}(\Theta)[t(n)] \cdot \mathrm{VAR}[O(v(n))]$
3. FO *uniform* $\mathrm{AC}(\Theta)[t(n)] \cdot \mathrm{WIDTH}[2^{O(v(n) \log n)}]$
4. $\mathrm{DTIME}[v(n)^{O(1)} + \log(t(n)]$- *unif.*$\mathrm{AC}(\Theta)[t(n)] \cdot \mathrm{WIDTH}[2^{O(v(n) \log n)}]$
5. $(\mathrm{CRAM}(\Theta)[t(n)] \cdot \mathrm{HARD}[2^{O(v(n) \log n)}]$

The proof of Theorem 4.6 is quite similar to the proof of Theorem 3.3. We simulate the computations as before, and just check that the width-hardware-variable resources needed are appropriate..

5 Uniformity: The Third Dimension

It has been understood for a while that non-uniformity corresponds in the descriptive setting to the addition of numeric predicates. Recall that a numeric predicate is a predicate such as $\leq$ or BIT that depends only on the numeric values of its arguments, not on any of the input predicates.

Fact 5.1 ([I87]) *A problem S is in Non-uniform* AC^0 *iff for some numeric predicate N, S is expressible in the language* $\mathrm{FO}(N)$.

It seems, after a fair amount of investigation and soul searching [I89b, BIS, L] that the "right" lowest level of uniformity corresponds to the numeric predicates $\leq, \mathrm{BIT}$. (This is equivalent to the set $\leq, +, \times$.) Once we have a little bit of computation, such as a majority quantifier or the deterministic transitive closure operator, all that is needed is $\leq$ and BIT is superfluous [BIS, I87].

The following theorem gives two examples of a very general phenomenon. The idea of capturing polynomial-time uniformity via the unary form of an EXPTIME complete problem is from [A].

Theorem 5.2 *Theorem 3.3 remains true in both the non-uniform and the polynomial-time uniform settings. More precisely, the classes mentioned in that theorem remain equal in the following cases:*

1. *When an arbitrary numeric predicate is added to first-order logic, and an arbitrary polynomial length "advice string" is given to the* CRAM *and to the circuits.*

2. *Let E be a numeric predicate that codes the unary version of an EXPTIME complete problem. Add E to the first-order languages* FO *and* IND*, add a table for E to the* CRAM*, and change, "DLOGTIME uniform" to "polynomial-time uniform".*

One feature of uniformity that we find amazing is that very low-level uniformity seems to suffice. Similarly, natural complete problems tend to remain complete via very low-level reductions such as fops. Why this is true is not completely clear. In part, the answer is the existence of universal Turing machines and thus universal complete problems. However, we feel that there is more to it than that. Here is a typical example:

Observation 5.3 *The following classes are all equal:*

1. $\mathrm{FO}[n^{O(1)}]$

2. *DLOGTIME uniform polynomial-size circuits*

3. *LOGSPACE uniform polynomial-size circuits*

4. *polynomial-time uniform polynomial-size circuits*

5. P

6 Variables that are Shorter than $\log n$ Bits

One would expect circuits of constant *width*, rather than depth, to be very weak, but the following two interesting characterizations tell us that this is not so:

Theorem 6.1 ([B89, BIS]) NC^1 *(DLOGTIME or* NC^1 *uniform) is exactly the set of languages recognized by DLOGTIME uniform boolean circuit families of* $O(1)$ *width and* $n^{O(1)}$ *depth.*

Theorem 6.2 ([CF]) *PSPACE is exactly the set of languages recognized by* NC^1 *uniform boolean circuit families of* $O(1)$ *width and* $2^{n^{O(1)}}$ *depth.*

Proof (sketch) It is clear that such circuits can be simulated in $PSPACE$. It remains to use these circuits to simulate a $PSPACE$ Turing machine, for which it suffices to solve the reachability problem on the exponential-size configuration graph. By the standard Savitch construction we get a circuit of polynomial depth and exponential size, and by standard tricks we can turn this into a boolean formula with fan-in two which is very uniform. (In particular, we can take the polynomial-length gate number and (in FO) recover the two Turing machine configurations which gave rise to the gate).

We then apply the construction of [B89] to get an exponential-length branching program of constant width (easily convertible into the desired circuit of constant width). Essentially (as explained in [BIS]) a gate number in the constant-width circuit encodes both a leaf node of the poly-depth circuit (and hence a pair of configurations of the original Turing machine) and an indication of which element of the group S_5 is to be computed by this level. Determining the latter means passing over the entire gate number (or from the root to the leaf of the poly-depth circuit), performing an operation in S_5 at every step, which is an NC^1-complete problem. □

Our goal in this section is to expand the descriptive complexity framework to encompass results such as Theorems 6.1 and 6.2. That is, we want to extend the previous notions in this paper to the situation where we have fewer than one $\log n$-bit variable, or equivalently circuit width less than polynomial.

By analogy with Section 4, we would like to define a class $\mathrm{FO}[t(n)] \cdot \mathrm{VAR}[v(n)]$ for $v(n) = o(1)$, which would be equivalent to uniform circuits of width $2^{O(v(n)\log n)}$ and depth $t(n)$. The difficulty in doing so is that the basic first-order variables of our formalism have $\log n$ bits, so that quantifying over one of them would appear to exceed the width bound. We have to prohibit explicit use of such variables, while retaining them in order to talk about the others.

It is absolutely necessary that our formulas, like the "bottleneck machines" of [CF], have access to a read-only clock. Thus we will allow the formulas within the quantifier block to have access to a variable t, which will indicate which iteration of the quantifier block we are currently in. As long as $t(n) = 2^{n^{O(1)}}$, t will always be only polynomially many bits, so we may access its individual bits by using the BIT predicate and a vector of ordinary variables. But we need to restrict ordinary quantifiers to maintain the width bound. Consider the following:

Definition 6.3 (Fewer than one variable) For $v(n) = o(1)$, and $t(n) = 2^{n^{O(1)}}$, an $\mathrm{FO}[t(n)] \cdot \mathrm{VAR}[v(n)]$ formula will have three sorts of variables, ordinary ones $x, y, z, \ldots$, ranging over the universe $\{1, \ldots, n\}$, limited ones $a, b, c, \ldots$, of $v(n) \log n$ bits each, and a variable t whose value is an integer, equal to 0 in the base formula M_0 and equal to j in the j^{th} quantifier block to the left of M_0. Both ordinary and extended variables may be quantified, while t is syntactically like a constant. Only ordinary variables may be used to access the input, but all may be used in the BIT predicate. Finally, the quantifier block B may not have an ordinary variable x that occurs freely in B. □

The purpose of the final restriction is to force the circuits obtained from these formulas, as in the proof of Theorem 3.3, to periodically have levels with only $2^{O(v(n) \log n)}$ gates. We would prefer to have a definition where *all* the levels were so bounded, but this definition produces a circuit class which is equivalent, allowing us to prove:

Theorem 6.4 $\mathrm{FO}[n^{O(1)}] \cdot \mathrm{VAR}[1/\log n] = $ *(DLOGTIME uniform)* NC^1.

Proof

($\subseteq$) : For every $t \leq t(n)$, because of the special condition, we know that $[B]^t M_0$ has free variables totaling $r = O(1)$ bits. Define $f(t)$ to be the 2^r-bit string defined by the truth value of this formula for all possible values of the r bits. The string $f(t)$ is a first-order function of $f(t-1)$, t, and the input predicates. Since $\mathrm{FO} \subseteq \mathrm{NC}^1$, this means that each bit of $f(t)$ is computable from these values via a bounded width, polynomial-size circuit. We can just string all of these $t(n)$ bounded-width circuits together one after the other to get the bounded-width, polynomial-size circuit for the original problem.

($\supseteq$) : By Theorem 6.1 we know that there are DLOGTIME-uniform, constant-width, polynomial-depth circuit families for any language in NC^1. Let 2^r be the width and $t(n)$ be the depth of the circuits for the problem in question. The values of the 2^r gates at level t of the circuit are determined in a DLOGTIME-definable way from t and the values at level $t-1$, and from the input. Recall that DLOGTIME is contained in FO [BIS]. Thus we can write this relationship in a first-order formula $\varphi(b_1, \ldots, b_r, G_{t-1})$. Here $G_{t-1}(b'_1, \ldots, b'_r)$ is a relation that codes the state of the 2^r gates at time $t-1$. Using a standard, syntactic trick, (Corollary 5.3 in [I86]), we can form a quantifier block B so that

$$G_t(b_1,\ldots,b_r) \quad\equiv\quad [B]G_{t-1}(b_1,\ldots,b_r)$$

Here, B may quantify some ordinary variables in order to look at the bits of the input that t tells it to, but no such ordinary variables will be left free. Thus, B is the quantifier block that we are looking for and we have that

$$G_t(b_1,\ldots,b_r) \quad\equiv\quad [B]^t G_0$$

as desired. □

Similarly, we show:

Theorem 6.5 $\mathrm{FO}[2^{n^{O(1)}}]\cdot\mathrm{VAR}[1/\log n] \;=\; \mathrm{PSPACE}$.

Proof

($\subseteq$) : This is immediate from previous results because $\mathrm{FO}[2^{n^{O(1)}}]\cdot\mathrm{VAR}[1/\log n] \subseteq \mathrm{FO}[2^{n^{O(1)}}] = \mathrm{PSPACE}$.

($\supseteq$) : We proceed much as in the similar case above, using the result of Theorem 6.2, except that our finite function simulating the effect of the t^{th} level of gates is now NC^1 computable rather than DLOGTIME computable from t and the input. However, we can now use Theorem 6.4 above to simulate the necessary NC^1 predicate via a $\mathrm{FO}[n^{O(1)}]\cdot\mathrm{VAR}[1/\log n]$ formula. We now just iterate this formula exponentially many times, getting the desired result. □

7 Conclusions

We have presented a very general, three-dimensional view of complexity. The dimensions are parallel time, amount of hardware, and amount of precomputation and correspond closely to quantifier-depth, number of variables, and complexity of numeric predicates, respectively. The tradeoffs between quantifier-depth, number of variables, and the complexity of numeric predicates are — to say the least — worthy of much future investigation.

Acknowledgments: Both authors gratefully acknowledge the support of the NSF Computer and Computation Theory program, through grants CCR-9207829 (Barrington) and CCR-9207797, CCR-9505446 (Immerman). Thanks also to Chi-Jen Lu, Ken Regan, and Zhi-Li Zhang for valuable discussions.

8 References

[A] E. Allender, "P-uniform circuit complexity," *JACM,* 36 (1989), 912 - 928.

[AG] E. Allender and V. Gore, "A uniform circuit lower bound for the permanent," *SIAM J. Comput.,* 23 (1994), 1026-1049.

[B89] D. Barrington, "Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1," *J. Comp. Syst. Sci.* 38 (1989), 150-164.

[B90] D.A.M. Barrington, "Extensions of an idea of McNaughton," *Math. Systems Theory,* 23 (1990), 147-164.

[B92] D.A.M. Barrington, "Quasipolynomial size circuit classes," *7th Structure in Complexity Theory Conference* (1992), 86-93.

[BIS] D. Barrington, N. Immerman, H. Straubing, "On uniformity within NC^1," *JCSS* 41(3 (1990), 274 - 306.

[BCH] P. Beame, S. Cook, H.J. Hoover, "Log depth circuits for division and related problems," *SIAM J. Comput.* 15(4) (1986), 994-1003.

[BT] R. Beigel and J. Tarui, "On ACC", *Computational Complexity* 4 (1994), 367-382.

[BFS] J. Boyar, G. S. Frandsen, and C. Sturtivant, "An arithmetic model of computation equivalent to threshold circuits", *Theoretical Computer Science* 93 (1992), 303-319.

[CF] Jin-Yi Cai and Merrick Furst, "PSPACE survives three-bit bottlenecks", *International Journal on Foundations of Computer Science*, 2(1) (1991) 67-76.

[CS] Jin-Yi Cai and D. Sivakumar, "The resolution of a Hartmanis conjecture", *36th Annual Symposium on Foundations of Computer Science* (1995), 362-371.

[FVB] G. S. Frandsen, M. Valence, and D. A. M. Barrington, "Some results on uniform arithmetic circuit complexity", *Math. Systems Theory* 27 (1994), 105-124.

[GKR] F. Green, J. Köbler, K. Regan, T. Schwentick, and J. Torán, "The power of the middle bit of a $\#P$ function", *J. Comp. Syst. Sci.*, to appear.

[H] Hong Jia-Wei, *Computation: Computability, Similarity, and Duality,* (New York: John Wiley & Sons, 1986).

[I86] N. Immerman, "Relational Queries Computable in Polynomial Time," *Information and Control* 68 (1986), 86-104.

[I87] N. Immerman, "Languages that capture complexity classes," *SIAM J. Comput.* 16(4) (1987), 760-778.

[I88] Neil Immerman, "Nondeterministic Space is Closed Under Complementation," *SIAM J. Comput.* 17(5) (1988), 935-938.

[I89] N. Immerman, "Descriptive and computational complexity," *Computational Complexity Theory,* ed. J. Hartmanis, *Proc. Symp. in Applied Math.,* 38, American Mathematical Society (1989), 75-91.

[I89b] N. Immerman, "Expressibility and parallel complexity," *SIAM J. Comput.* 18 (1989), 625-638.

[I91] N. Immerman, "DSPACE[n^k] = VAR[k+1]," *6th Structure in Complexity Theory Conf.* (1991), 334-340.

[IL] N. Immerman, S. Landau, "The complexity of iterated multiplication," *Information and Computation* 116(1) (1995), 103-116.

[KL] R. M. Karp and R. J. Lipton, "Turing machines that take advice", *Enseign. Math.* 28 (1982), 191-209.

[KV] Phokion Kolaitis and Jouko Väänänen, "Generalized Quantifiers and Pebble Games on Finite structures, *Annals of Pure and Applied Logic*, 74(1) (1995), 23–75.

[L] S. Lindell, "A purely logical characterization of circuit uniformity," *7th Structure in Complexity Theory Conf.* (1992), 185-192.

[MP] J.A. Makowsky and Y.B. Pnueli, "Computable quantifiers and logics over finite structures," to appear in *Quantifiers: Generalizations, Extensions and Variants of Elementary Logic* (Kluwer Academic Publishers, 1993).

[O] Mitsunori Ogihara, "Sparse P-hard sets yield space-efficient algorithms", *36th Annual Symposium on Foundations of Computer Science* (1995), 354-361.

[Re] J. Reif, "On threshold circuits and polynomial computation," *2nd Structure in Complexity Theory Conf.* (1987), 118-123.

[Ru] L. Ruzzo, "On uniform circuit complexity," *J. Comp. Sys. Sci.,* 21(2) (1981), 365-383.

[SV] Larry Stockmeyer and Uzi Vishkin, "Simulation of Parallel Random Access Machines by Circuits," *SIAM J. of Comp.* 13(2) (1984), 409-422.

[V] L.G. Valiant, "Reducibility by algebraic projections," *L'Enseignement mathématique,* 28 (1982), 253-268.

[vM] D. van Melkebeek, "Reducing P to a sparse set using a constant number of queries collapses P to L", *Eleventh Annual IEEE Conference on Computational Complexity (formerly known as STRUCTURES)* (1996), to appear.

[Y] A. C. C. Yao, "On ACC and threshold circuits", *Proc. 31st Ann. IEEE Symp. Foundations of Computer Science* (1990), 619-627.