A Purely Logical Characterization of Circuit Uniformity
(*Extended Draft*)

Steven Lindell †

Haverford College
Haverford, PA 19041-1392

Abstract

Utilizing the connection between uniform constant-depth circuits and first-order logic with numerical predicates, we are able to provide a purely logical characterization of uniformity based on the intrinsic properties of these predicates. By requiring a numerical predicate R *to satisfy a natural extensibility condition — that it can be translated to a polynomially magnified domain based on tuple constructions — we show that* R *must already be elementarily definable from* < *and* bit *(both of which satisfy the extensibility condition). Our answer is motivated by, and coincides with,* DLOGTIME *uniformity.*

0. Introduction

This paper explores a purely logical description of circuit uniformity, justifying the resulting class of *DLOGTIME*-uniform bounded-depth circuits. Relying on results in [BIS], these will be the queries which are expressible in first-order logic on finite ordered structures which contain the bit predicate, abbreviated $FO(<, bit)$.

By equating uniform families of circuits with alternating Turing-machines, Ruzzo [Ruz], shows that logarithmic-depth circuits of bounded fan-in gates, NC^1, are a reasonable model of parallel computation, relatively insensitive to various uniformity definitions.

Uniform families of bounded-depth circuits, AC^0, and their connection with first-order logic are explored in [G&L]. As [Bar] notes, they observed that the equivalence between bounded-depth circuits and first-order formulas with auxiliary relations holds at virtually any uniformity level. Corollary 6 in [Bar] makes clear this correspondence and refines their results to read:

"*Let C be any class of functions containing the log-time computable functions and closed under composition. Then a language is in C-uniform* AC^0 *iff it can be expressed by a first-order formula using numerical predicates in C.*"

The class of *numerical predicates* referred to indicates precisely those relations whose interpretation is dependent solely on the size of the input. In particular, if R is a numerical predicate, then for every structure of size n, R_n is equal to a fixed k-ary relation on $\{0, 1, \ldots, n-1\}$. The proof of the quoted result relies on a fundamental result in [BIS] connecting a very restrictive notion of circuit uniformity with first-order logic at an arguably very low level.

Overview of paper

Section 1 provides a brief review of first-order logic, mostly by examples. Included is the important concept of viewing a binary string as a finite structure, and its augmentation by the special bit predicate.

Section 2 explains how to view circuit uniformity from a relational query perspective, by illustrating the equivalence between *DLOGTIME*-uniform bounded-depth circuits and expressibility in first-order logic with < and *bit*. This section also mentions how $FO(<, bit)$ is equivalent to constant-time computation on a parallel random-access machine, the *WRAM*, where each processor is permitted a special shift operation.

Further reasons to consider $FO(<, bit)$ from a logical query perspective are given in Section 3, in which the isomorphism problem for binary string structures is discussed, along with structures which use arithmetic as their auxiliary relations.

These motivational subjects are summarized in Section 4, and the definition of a first-order translation is explained in Section 5. Section 6 gives examples of extensibility for the relations +, <, and *bit*.

Section 7 gives the precise definition of what it means for an auxiliary numerical predicate to be uniformly extensible, followed by the main result and sketch of its proof in section 8. The paper's conclusion, section 9, is followed by open problems.

† email: S_LINDELL@acc.haverford.edu. Research completed while the author was on leave in residence 1990-91, and also partially supported by NSF grant CCR-9003356.

Appeared in: Proceedings of the 7th Annual IEEE Conference on Structure in Complexity Theory, (1992), pp. 185-192.

1. Brief review of first-order logic

First-order logic on finite structures provides a machine independent model of parallel complexity theory. This approach views combinatorial problems as *queries* on classes of *finite structures*. One of the most familiar examples is the class of digraphs—all structures having a binary edge relation E over a finite domain of vertices V:

$$G = \langle V, E \rangle \qquad |V| = n \qquad E \subseteq V^2$$

The problem of determining if a graph is *simple* (no self-loops, all edges undirected) is an example of a graph property, or Boolean query. Another example is the problem of determining if the edge relation constitutes a total linear *order* of the vertices. Both of these properties are expressible as first-order sentences. Simplicity is:

$$G = \langle V, E \rangle \text{ is } \textit{simple} \text{ iff } G \models \theta \qquad \text{where}$$

$$\theta \equiv \neg(\exists x)[E(x, x)] \wedge (\forall y)(\forall z)[E(y, z) \rightarrow E(z, y)]$$

The first part of θ says that no vertex has an edge to itself, and the second part says that if there is an edge from one vertex to another, then there must be an edge going the other direction. The term *first-order* refers to the fact that the only allowable quantification is the binding of individual variables which range over the domain.

Using more suggestive notation for orderedness:

$$\mathbf{B} = \langle A, < \rangle \text{ is } \textit{ordered} \text{ iff } \mathbf{B} \models \psi$$

where we can express ψ as the conjunction of the following axioms (all variables universally quantified):

$\neg(x < x)$	(irreflexivity)
$(x \neq y) \rightarrow [(x < y) \vee (y < x)]$	(totality)
$[(x < y) \wedge (y < z)] \rightarrow (x < z)$	(transitivity)

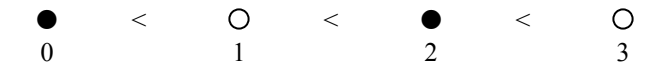
In general, a *finite relational structure*,

$$\mathbf{A} = \langle A, R_1, \ldots, R_k \rangle,$$

consists of a finite set A called the *domain*, together with *relations*, each of a specified arity on A. Formulas in first-order logic for these structures permit: individual variables interpreted as ranging over the domain; predicate symbols for each relation R_i and equality (=); the Boolean connectives ∧, ∨, and ¬; and first-order quantification of the variables. We denote the class of all such first-order queries *FO*. For further background, see [End].

Binary string structures

The most important example, from a computational point of view, is the class of binary string structures. Each such structure is of the form $\mathbf{B} = \langle B, <, U \rangle$, where $<$ is a linear order on B, and U is a unary relation on B. Interpreted as a binary string, $<$ orders the positions in the string from left to right, and U indicates the positions of the **1**'s and **0**'s by true and false, respectively. For instance, the binary string **1010** is represented by the structure $\langle \{0, 1, 2, 3\}, <, \{0, 2\} \rangle$, with $0 < 1 < 2 < 3$.



The closed circles indicate where U is true, and the open circles where U is false.

Since on an ordered structure each element of the domain can be thought of as corresponding to a number which is it's distance from the smallest element, we will henceforth use the set $\{0, 1, \ldots, n-1\}$ for the domain of an ordered structure, with the obvious intention that $0 < 1 < \ldots < n-1$ under the ordering of the structure.

Adding bit to ordered structures

To directly express resource-bounded computation on machine models, it is necessary to work on ordered structures — those which always include a total linear ordering of the domain. Furthermore, a special binary predicate called *bit* is useful in order to capture notions of constant-time parallel computability. For a domain element i, we will be able to determine the location of **1**'s and **0**'s in its binary representation:

$$\textit{bit}(i, j)$$
$$\Leftrightarrow$$
the jth position in the binary representation of i is a **1**.

For instance, $\textit{bit}(5, 1)$ is false, since $5 = (\mathbf{101})_2$, and there's a **0** in the 1st position (the rightmost bit is treated as the 0th position). Adding *bit* to the ordering results in structures of the form

$$\langle A, <, \textit{bit}, R_1, \ldots, R_k \rangle,$$

and we distinguish the augmented class of first-order queries by the notation:

$$FO(<, \textit{bit}).$$

2. Connections with parallel complexity

The central importance of *bit* lies in its connection with parallel computation, both in terms of uniform families of bounded-depth circuits and parallel random-access machines.

Uniform constant-depth circuits

Given a bounded-depth family of polynomial-size circuits C_n, $n = 1, 2, \ldots$, with $|C_n| \leq n^d$ (unbounded fan-in, AC^0), consider the complexity of the map:

$$n \xrightarrow{f} C_n$$

The inputs to C_n consist of binary strings of size n, each bit in the string representing the value *true* or *false*. We should imagine the nodes of the circuit as being numbered in some fashion from 1 to n^d, each node having the property that it is a certain kind of gate (**and**, **or**) or else a leaf (one of the input bits or its negation). *DLOGTIME* uniformity means that given a pair of nodes in C_n, we can compute whether or not they are connected in time $O(\log n)$ on a deterministic Turing-machine with a random-access tape (and given a single node, we can tell what kind of node it is — either its gate type or its location relative to the input string). Since each node can be described using $d \cdot \log_2 n$ symbols, we can think of this as a kind of linear time uniformity in the actual size of the input. This very restrictive notion of circuit uniformity relates to first-order logic in a beautiful way [BIS]:

Theorem: *DLOGTIME-uniform* $AC^0 = FO(<, bit)$.

Parallel random-access machines

Another model of parallel computation is a machine with polynomially many processors, each with simultaneous read/write random-access to a global memory (write conflicts are resolved by giving priority to the lowest numbered processor). The input is stored initially in the first n positions of the memory, and each processor has a special shift instruction allowing it move a local register over by up to $\log_2 n$ bits. In another beautiful result, constant-time parallel computability on this model, denoted $WRAM(O(1), shift)$, is connected to first-order logic [Im3]:

Theorem: $WRAM(O(1), shift) = FO(<, bit)$.

3. Connections with query definability

We wish to bring to the reader's attention two further facts that indicate the importance of the bit predicate from a query definability point of view.

The isomorphism problem for binary strings

One connection with first-order query definability comes from examining the structure isomorphism problem, discussed extensively in [Lin]. Essentially, the *isomorphism query* asks if two given structures (of the same type) are isomorphic. In the case of graphs, we can formulate this question as a Boolean query *ISO* which corresponds to graph isomorphism:

$$\langle V, E, E' \rangle \models ISO$$
$$\Leftrightarrow$$
$$\langle V, E \rangle \cong \langle V, E' \rangle.$$

In the case of binary string structures, this becomes:

$$\langle B, <, U, <', U' \rangle \models ISO$$
$$\Leftrightarrow$$
$$\langle B, <, U \rangle \cong \langle B, <', U' \rangle$$

where we tacitly assume that both $<$ and $<'$ are total linear orders of the same underlying domain. The following theorems are proved in [Lin]:

Theorem: *The isomorphism problem for binary strings is not first-order definable.*

On the other hand, if we add a bit predicate to each of the orders in a corresponding fashion, we obtain what essentially looks like the same problem:

$$\langle B, <, bit, U, <', bit', U' \rangle \models ISO$$
$$\Leftrightarrow$$
$$\langle B, <, bit, U \rangle \cong \langle B, <', bit', U' \rangle$$
$$\Leftrightarrow$$
$$\langle B, <, U \rangle \cong \langle B, <', U' \rangle$$

Except that in this case, the following is actually true [Lin]:

Theorem: *The isomorphism problem for binary strings with a bit predicate is first-order definable.*

This result can be seen as showing that string isomorphism (with *bit*) is the simplest possible non-trivial structure isomorphism problem (but it should be noted that its proof is far from trivial).

Arithmetic on finite structures

An appealing correspondence between the relatively obscure bit predicate and the comparatively more familiar functions of ordinary arithmetic can be made. Although *bit* is apparently a way to encode/decode numbers in binary based on their position in an ordering, and hence seems to be intimately connected with binary representation, it is a surprising but little-known fact that elementary definability with {<, *bit*} is equivalent (i.e. up to a first-order factor) to elementary definability with the arithmetical partial functions {+, *, ^}.

Consider the usual operations of arithmetic given by the ternary relations

$$\begin{array}{ll} Plus(a, b, c) \Leftrightarrow a + b = c & \text{(addition)} \\ Times(a, b, c) \Leftrightarrow a * b = c & \text{(multiplication)} \\ Exp(a, b, c) \Leftrightarrow a \,\hat{}\, b = c & \text{(exponentiation)} \end{array}$$

restricted so that a, b, c range over a finite domain $\{0, 1, \ldots, n-1\}$. Not only are the arguments required to range over the domain, but the functions are not permitted to overflow. Define an *arithmetical finite structure* as one of the form:

$$\langle \{0, 1, \ldots, n-1\}, +, *, \hat{}, R_1, \ldots, R_k \rangle,$$

where we assume +, *, ^ satisfy the usual axioms of arithmetic, and let

$$FO(+, *, \hat{})$$

denote the class of first-order queries augmented by these arithmetical predicates. We claim that $FO(+, *, \hat{})$ essentially coincides with $FO(<, bit)$.

Theorem: *On arithmetical finite structures*,

$$FO(<, bit) = FO(+, *, \hat{})$$

where the numerical predicates <, *bit*, +, *, ^ *are all assumed to satisfy their standard axioms over the domain*.

One direction of this equivalence, that < and *bit* are elementarily definable from +, *, and ^, is quite easy. The ordering can be redefined from addition:

$$i < j \Leftrightarrow i \neq j \wedge (\exists k)[i + k = j]$$

The additive and multiplicative identities define zero and one:

$(\forall a)[a + 0 = a]$ defines zero

$(\forall a)[a * 1 = a]$ defines one

Mixed fraction representation of i / j, given by $i = k * j + l$ with $0 \leq l < j$ defines integer division. So

$$\begin{array}{ll} i \text{ div } j = k & \text{the quotient of } i / j \\ i \bmod j = l & \text{the remainder of } i / j \end{array}$$

define the *div* and *mod* functions as quotient and remainder, respectively. Combined with the important definition of two, 1 + 1 = 2, we get

$$bit(i, j) \Leftrightarrow (i \text{ div } 2\hat{}j) \bmod 2 = 1$$

where (i div $2\hat{}j$) removes the j least significant bits of i, and mod 2 examines the remaining least significant bit. If $2\hat{}j$ doesn't exist (i.e. overflows the domain of the structure), then the formula fails to be satisfied and $bit(i, j) = 0$ as expected.

The reverse direction, of defining +, *, and ^ from < and *bit*, is substantially harder. In this extended draft, we will simply confine our remarks to mention that the key idea in the proof consists of converting the arguments into binary (using *bit*) and then performing some fairly complicated algorithms for (log n)-bit binary arithmetic.

4. What is so special about *bit*?

To summarize, we have shown the reader three compelling *extrinsic* reasons for including the numerical predicates < and *bit* in first-order definability on finite structures.

• $FO(<, bit)$ captures two equivalent notions of uniform constant-time parallel computability:

$$FO(<, bit) = WRAM(O(1), shift)$$
$$FO(<, bit) = DLOGTIME\text{-uniform } AC^0.$$

• The isomorphism problem for binary string structures with the bit predicate is first-order definable.

$$\langle B, <, bit, U \rangle \cong \langle B, <', bit', U' \rangle$$
$$\Leftrightarrow$$
$$\langle B, <, bit, U, <', bit', U' \rangle \models ISO$$
where
$$ISO \in FO$$

• First-order definability with < and *bit* is the same as first-order definability with arithmetic:

$$FO(<, bit) = FO(+, *, \hat{})$$

These correspondences tell us that *bit* is special, but from a purely logical viewpoint don't tell us *what* is special about it. In particular, one might wonder if there are further predicates to be discovered that could augment first-order definability:

$$FO(<, bit, ?)$$

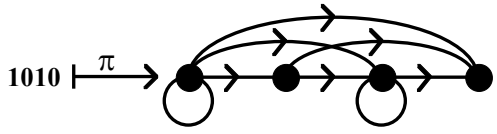
The aim of this paper is to show that there is nothing else! That is, no numerical predicate which satisfies a certain kind of uniform extensibility condition provides a proper extension of $FO(<, bit)$ on finite structures. The precise answer comes from a close examination of the logically *intrinsic* properties of < and *bit*. And to do this, we turn to logical reductions between problems.

5. First-order translations

Reductions have been a key tool in understanding resource-bounded computation. Many well-studied complexity classes with nice closure properties, such as *PSPACE*, *NP*, *P*, *NL*, *L*, and *ALOGTIME* all contain problems which are demonstrably hardest in the sense that all other problems in their class can be *reduced* to them via some very simple sort of conversion process, such as one using logspace transducers. These hardest problems are called *complete*, and the phenomenon of completeness has played an important role in understanding the relative complexity of combinatorial problems.

To pick just one example, *GAP*, the graph accessibility problem of determining if there is a directed path from *a* to *b* in a graph, is complete for *NL* (= co-*NL* [Im1]) under first-order reductions [Im2]. These are much stricter than the usual logspace reductions and furthermore are provably weaker.

First-order reductions correspond to the logical notion of *translating* a structure in a certain vocabulary to another structure in a potentially different vocabulary. For example, to translate binary strings into graphs, one could use the following idea:



$$\pi$$

$$\langle B, <, U\rangle \longrightarrow \langle V, E\rangle$$

and write it formally as the formulas:

$$\pi_\forall(x) \equiv true \qquad \text{(defines } V\text{)}$$
$$\pi_E(y, z) \equiv (y < z) \vee [(y = z) \wedge U(y)] \qquad \text{(defines } E\text{)}$$

That is to say, the image of a binary string under this translation is the graph whose vertex set is the same as the positions of the string, and whose edge relation lies between a strict and non-strict linear order (in a set-theoretic sense)

$$(<) \subseteq E \subseteq (\leq)$$

such that E has a self-loop at x iff the position corresponding to x had a 1 located there. Curiously enough, the inverse problem of converting an arbitrary graph into a binary string

$$\langle V, E\rangle \longrightarrow \langle B, <, U\rangle$$

is the graph canonization problem, which appears to be very difficult (it is in the polynomial-hierarchy, *PH*, but not known to be in *P*).

To fully qualify as a combinatorial reducibility for complexity theory (in order to retain the existence of complete problems), we must provide a mechanism for allowing translations to effect a polynomial increase in problem size (as measured by the cardinality of the domain). A simple and natural way to do this syntactically in logic is via *tuple* constructions, in which a fixed number of elements are combined into a vector of length m. If the original domain had n elements, the set of all possible m-vectors forming the new domain has n^m elements. The other relations in the translation are defined on this set of m-tuples. The precise generalization is:

Definition: Given signatures composed of predicate symbols $\sigma = \{S_1, \ldots, S_k\}$ and $\tau = \{T_1, \ldots, T_l\}$, let $\mathbf{\Sigma}$ be the set of all finite structures of type σ and let $\mathbf{T}$ be the set of all finite structures of type τ. A function $\pi: \mathbf{\Sigma} \to \mathbf{T}$ is called a *first-order translation* if π can be expressed by a sequence of first-order formulas in the signature σ

$$\pi = \langle \pi_\forall(\boldsymbol{x}), \pi_{T_1}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{b_1}), \ldots, \pi_{T_k}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{b_k})\rangle$$

where **boldface** $\boldsymbol{x} = \langle x_1, \ldots, x_m\rangle$ is an m-tuple, m is called the *magnification factor*, and b_i is the arity of T_i, $1 \leq i \leq k$.

So given a structure $\mathbf{A}$ of type σ, $\pi(\mathbf{A})$ will be a structure of type τ, with domain $[\pi_\forall(\boldsymbol{x})]^{\mathbf{A}}$ (typically $\pi_\forall(\boldsymbol{x})$ is just the always true formula, making the domain of the image to be all m-tuples from the domain of $\mathbf{A}$). Each relation, T_i, of $\pi(\mathbf{A})$ is given by evaluating the corresponding formula $[\pi_{T_i}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{b_i})]^{\mathbf{A}}$. This is a direct generalization of the classical notion of an *interpretation* as defined in [End] — translations in which the magnification factor is absent (i.e. 1). But on finite structures it becomes necessary to include arbitrarily large magnification factors to permit a polynomial size increase in domain cardinality.

6. Examples of auxiliary relations

Now, if we take literally the fact that each and every relation on $\pi(\mathbf{A})$, including equality, must have a first-order definition in the translation, we are led to the inevitable conclusion that if τ contains a symbol for a numerical predicate R, then π must contain a formula π_R which defines it. If we want to translate binary string problems — the standard vocabulary for computational complexity — then we must find a way to define = and < (and *bit*) on polynomially magnified domains. The remainder of this section is devoted to demonstrating how simple the required formulas are, and in doing so motivate the definition in the next section of what it means for a numerical predicate to be uniformly extensible.

Equality

In defining the structure $\pi(\mathbf{A})$ in terms of tuples from **A**, it is easy to see that equality of the resulting elements is quite simple to determine. For instance, if $m = 2$:

$$\langle x_1, x_2\rangle = \langle y_1, y_2\rangle \;\Leftrightarrow\; x_1 = y_1 \wedge x_2 = y_2$$

The cases $m > 2$ are obvious too. In particular, the formula for $\pi_{=}(\boldsymbol{x}, \boldsymbol{y})$ is always propositional.

Ordering

Now, let us consider translations among finite ordered structures. This means that there is a binary predicate symbol < which is both a member of σ and τ and whose interpretation always satisfies the axioms for a strict linear order. In particular, there must be a formula $\pi_{<}(\boldsymbol{x}, \boldsymbol{y})$ which defines a strict linear order on m-tuples. Clearly, $\pi_{<}$ will depend on the order of individual elements, and nothing else. In fact, lexicographic order is an obvious choice. For $m = 2$, this means that $\pi_{<}(\boldsymbol{x}, \boldsymbol{y})$ defines:

$$\langle x_1, x_2\rangle < \langle y_1, y_2\rangle \;\Leftrightarrow\; x_1 < y_1 \vee x_1 = y_1 \wedge x_2 < y_2$$

For $m > 2$ use the same standard technique. Notice that this expression still only requires propositional logic, and that the number of occurrences of < in the formula for $\pi_{<}$ is m. So we see that the ordering relation also satisfies a very simple sort of polynomial extensibility. For convenience from now on, we will assume that further numerical predicates will "sit" on the ordered domain whose elements are named $\{0, 1, \ldots, n-1\}$.

Bit

In expressing polynomial extensions of the bit predicate, we proceed by a sequence of constructions. Begin by defining addition as the quintary predicate:

$$Add(a, b, c_i, s, c_o) \;\Leftrightarrow\; a + b + c_i = s + c_o$$

where $a, b, s \in \{0, 1, \ldots, n-1\}$ are the *addends* and the *sum* respectively, and $c_i,\ c_o \in \{0, 1\}$ are the *carry in* and *carry out*, respectively. Then for $m = 2$, (it is not necessary to pair the *carry in* or *carry out*) we can have π_{Add} express the addition problem:

$$\begin{array}{r} c \quad c_i \\ \langle a_1, a_2\rangle \\ +\,\langle b_1, b_2\rangle \\ \hline = \; c_o\,\langle s_1, s_2\rangle \end{array}$$

$$\Leftrightarrow$$

$$[Add(a_2, b_2, c_i, s_2, 0) \wedge Add(a_1, b_1, 0, s_1, c_o)] \vee [Add(a_2, b_2, c_i, s_2, 1) \wedge Add(a_1, b_1, 1, s_1, c_o)]$$

which covers the two cases for the intermediate carry $c = 0, 1$. Notice that we have allowed a constant (0 or 1) to be substituted for some of the arguments. This is because 0 and 1 are constants, elementarily definable from <, which was previously shown to be uniformly extensible (the precise definition appears in the next section). Addition on tuples of length $m > 2$ are also definable in the same manner, by taking into account all possible intermediate carries. Again, only propositional logic is required.

Next, for $n = 2^k$, let us show that the constant $\log_2 n$ is logically uniform for $m = 2$:

$$\langle z_1, z_2\rangle = \log_2 n^2 \;\Leftrightarrow\; z_1 = 0 \wedge z_2 = 2{\cdot}\log_2 n$$

This just says that the logarithm of the magnified domain of pairs, which has size n^2, is twice the logarithm of the original domain of size n. Note that we permit an elementary function (in this case *twice*) defined in terms of a previously extensible relation (*Add*) to appear (as $2{\cdot}\log_2 n$).

Finally, to express extensions the *bit* predicate, consider only powers of 2, $n = 2^k$ for simplicity:

$$bit(\langle x_1, x_2\rangle, \langle 0, y_2\rangle) \Leftrightarrow [(y_2 < \log_2 n) \wedge bit(x_2, y_2)] \vee [(y_2 \geq \log_2 n) \wedge bit(x_1, y_2 - \log_2 n)]$$

Since we are dealing with exact powers of 2, this formula just says that the binary expansion of a pair is the con-

catenation of the binary expansions of the individual elements. Notice again that we make use of a previously definable function, in this case $\log_2 n$, and subtraction (which is elementarily definable in terms of addition). It should be clear how to extend this definition to cover when $m > 2$, provided n is still a power of two. For arbitrary n, it would probably be easier to simply show that +, *, and ^ are uniformly extensible, and then derive the extension of *bit* from that.

7. Uniform extensibility of predicates

With our motivational sketches now complete, we turn to actually define what a uniformly extensible predicate is, consistent with our previous examples of =, <, and *bit*.

Definition: The numerical predicate $R(y_1,\ldots, y_k)$ is said to be *uniformly extensible*, if for $m > 1$, there exists a first-order formula ϕ such that

$$R(\langle y_{1,1},\ldots, y_{1,m}\rangle, \ldots,\langle y_{k,1},\ldots, y_{k,m}\rangle)$$
$$\Leftrightarrow$$
$$\phi(R;\ \{y_{i,j} : i = 1,\ldots, k; j = 1, \ldots, m\}).$$

Letting **boldface** variables represent m-tuples, e.g. $\boldsymbol{x} = \langle x_1,\ldots, x_m\rangle$, we can abbreviate this as:

$$R(\boldsymbol{y}_1,\ldots, \boldsymbol{y}_k) \Leftrightarrow \phi(R;\ \{y_{i,j} : i = 1,\ldots, k; j = 1,\ldots, m\})$$

Furthermore, every occurrence of R in ϕ is required to be of the form

$$R(f_1,\ldots, f_k),$$

where each argument f is an elementary function defined by a first-order formula θ,

$$f(\{y_{ij} : i = 1,\ldots, k; j = 1, \ldots, m\}) = z$$
$$\Leftrightarrow$$
$$\theta(\{y_{ij} : i = 1,\ldots, k; j = 1, \ldots, m\}, z).$$

In addition, ϕ (and hence any θ) may include other relations already known to be uniformly extensible (see prior examples), but θ is not permitted to contain R.

In simple terms, the most important feature of our definition is that it requires R_{n^m} to be definable using only finitely many recursive calls to R_n with non-recursive arguments. The ϕ allowed in our definition seems to be a generalization of a first-order projection translation [Im2, see also I&L, Def. 3.7].

8. Main result

Theorem: *If* R *is a uniformly extensible numerical predicate, then* R *is elementarily definable from* < *and* bit. *I.e. there exists a first-order formula* ψ *in the signature of* {=, <, *bit*} *such that*:

$$R(x_1,\ldots, x_k) \Leftrightarrow \psi(=, <, bit;\ x_1,\ldots, x_k)$$

N.B. ψ does *not* contain R (cf. ϕ in prior definition).

Proof (sketch): Technically the proof is by induction, where we assume that any relations already known to be uniformly extensible satisfy the theorem, and hence are in $FO(<, bit)$. We sketch the idea here for $m = 2$, though the proof scales to larger m.

In [BIS], it is shown that acceptance of a bounded-alternation logarithmic-time random-access Turing-machine is equivalent to first-order definability with < and *bit*. So it suffices to show that for any given set of arguments $x_1,\ldots, x_k$ to R, we can calculate the truth or falsity of $R(x_1,\ldots, x_k)$ in *LH*, the logtime hierarchy. This will directly imply that R is expressible in $FO(<, bit)$.

Suppose we are given the arguments $x_1,\ldots, x_k$ to R_n in binary. Each argument x_i has bit size $\log_2 n$ (for simplicity, assume that n is an exact power of 2). Our solution is presented as an *LH* algorithm for the Boolean evaluation of $R_n(x_1,\ldots, x_k)$.

1. Unfold the recursion: By uniform extensibility, R_n has a definition given by ϕ in terms of a fixed number of recursive calls to $R_{\sqrt{n}}$. The number of calls, l, is a syntactical property equal to the number of occurrences of R in ϕ. If we unfold this recursion entirely we obtain an l-ary tree of depth $\log_2 \log_2 n$ and size about $(\log_2 n)^{(\log_2 l)}$ with root $R(x_1,\ldots, x_k)$ and leaves of the form $R_2(b_1,\ldots, b_k)$, where the b_i are 1-bit arguments. An internal node of the tree looks like $R_n(y_1,\ldots, y_k)$ where each argument y_i is $\log_2 n'$ bits long. It is parent to l children, each of which look like $R_{\sqrt{n}}(f_1,\ldots, f_k)$, where the f_i are $\log_2 \sqrt{n'} = (\frac{1}{2})\log_2 n'$ bits long, and each may depend on the $2k$ arguments $y_i = \langle y_{i,1}, y_{i,2}\rangle$, $1 \le i \le k$, given by literally chopping each y_i in half. The argument dependencies from parent to child are given by the particular θ's in a first-order computable way, and the Boolean dependency of the parent on its children is first-order computable via ϕ.

2. Top-down Boolean evaluation: Evaluate the tree in $\log_2 l$ phases by breaking it top-down into trees of strictly logarithmic size. Guess the Boolean values of the top $\log_2 n$ nodes, and verify correctness locally by checking that each node follows from its children via ϕ (see the next step for details). Then, decide which of the $O(\log n)$ leaves of this sub-tree to isolate (with universal alterna-

tion), and using it as a new root repeat this process until reaching the bottom. This uses a bounded number of alternations $O(\log_2 l)$ precisely because the sub-trees are *independent*. At the leaves, it is necessary to have a (finite) table for checking R_2, which acts like a basis for the recursion.

3. Bottom-up argument determination: To verify that a parent's value follows from its children's values involves checking a first-order formula (ϕ) which may also contain other (first-order) relations. Those relations may depend on particular arguments, and so the problem remains to compute the arguments that appear at each node. If a parent has arguments of bit size b, then each of its l children will have arguments of bit size $b/2$ (for a magnification factor of $m = 2$). The arguments at any node are determined by guessing *all* the arguments on the path from the desired node to the root, which a quick computation reveals is no more than $2k \cdot \log_2 n$ bits. Verify this guess by checking that the arguments of a child follow from the arguments of its parent via the appropriate θ formulas, and make sure that the guess is consistent with the original group of arguments at the root, $x_1, \ldots, x_k$. There is a lot of bookkeeping going on here, especially in deciding which θ to choose at a given node, based on its position in the tree.

9. Conclusions

We have given a purely logical definition of circuit uniformity based solely on tuples and relations. A predicate was called uniformly extensible if it could be "passed-through" a translation in a very simple way. This is a natural requirement for studying reducibilities among queries on finite structures which include these predicates. It was construed to permit only those relations which when interpreted on tuples, can be defined in terms of themselves (on the individual elements) using a slight generalization of first-order projective translations. We have demonstrated that first-order logic augmented by all uniformly extensible relations captures $FO(<, bit)$ definability exactly, which is equivalent to *DLOGTIME*-uniform constant-depth circuits.

10. Open Problems

A. Can the main result be extended to a *functional* (as opposed to relational) setting? [I can get $O(\log^* n)$-time computability, but not $O(1)$]. How about allowing arbitrary first-order translations for ϕ (repealing the finite call condition)? [It is at most $O(\log \log n)$-time, which is just the depth of the tree].

B. Even though there cannot be any complete problems for $FO(<, bit)$, due to the strict depth hierarchy in [Sip], is there any other way to formalize the notion of a "hardest" problem for FO(<, *bit*)? The binary string *canonization query* (cf. the isomorphism query in section 4) discussed in [Lin] is intuitively "hard" yet elusive in its comparison with other problems.

C. Is *modular exponentiation* a uniformly extensible relation under even the more lenient requirement of a first-order translation (see above)? [$mexp(a, b, c, d)$ iff $(a \wedge b \equiv c) \bmod d$]. This would have important consequences in analyzing the uniformity of NC^1 circuits for binary division [I&L].

Acknowledgments

Special thanks to Eric Allender for helping me to understand *DLOGTIME* uniformity. Also, discussions with Neil Immerman, Dave Barrington, Howard Straubing, Greg McColm, and Larry Ruzzo were much appreciated and very helpful. Thanks to Scott Weinstein for suggesting the philosopher's terminology of "uniformly extensible". Particular appreciation goes to my wife, Suzanne, for her patience.

References

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

[BIS] D.A.M. Barrington, N. Immerman, H. Straubing, "On Uniformity within NC^1," Journal of Computer and System Sciences, vol. 41, no. 3 (Dec. 1990) 274-306.

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

[G&L] Y. Gurevich, H.R. Lewis, "A Logic for Constant-Depth Circuits," Information and Control **61**, 65-74 (1984).

[Im1] N. Immerman, "Nondeterministic Space is Closed under Complementation," SIAM J. of Computing 17:5 (Oct. 1988), 935-938.

[Im2] N. Immerman, "Languages that Capture Complexity Classes," SIAM J. of Computing, vol. 16, 1987, pp. 760-778.

[Im3] N. Immerman, "Expressibility and Parallel Complexity," SIAM Journal of Computing, vol. 18 no. 3, June 1989, pp. 625-638.

[I&L] N. Immerman, S. Landau, "The Complexity of Iterated Multiplication," Fourth Annual Structure in Complexity Theory Symp. (1989), 104-111.

[Lin] S. Lindell, "The Invariant Problem for Binary String Structures and the Parallel Complexity Theory of Queries," to appear in The Journal of Computer and System Sciences, 1992.

[Ruz] W.L. Ruzzo "On Uniform Circuit Complexity," J. Comp. Sys. Sci., 21:2 (1981) 365-383.

[Sip] M. Sipser, "Borel Sets and Circuit Complexity," STOC 1983, 61-69.