Is Multiplication Harder than Addition? Arithmetical Definability over Finite Structures

Troy Lee

December 11, 2001

Master's Thesis
Institute for Logic, Language, and Computation

Under the Supervision of
Harry Buhrman

Acknowledgements

This thesis owes a lot to David Mix Barrington, as it was a conversation with him that brought the problems discussed here to my attention. It was also series of lectures given by him the year before which introduced me to the field of complexity in the first place. I would like to thank Neil Immerman and David Mix Barrington for checking the proof of Thm (7.1.1).

Harry Buhrman very generously gave his time to be my supervisor, to give exposition improving comments on earlier drafts of the thesis, and to be a formidable ping-pong opponent. He expeditiously provided me with the resources and funding that made finishing the thesis much easier and enjoyable. Thanks, Harry!

I would like to thank Johan van Benthem and Leen Torenvliet for giving their time to sit on my committee, and Alan Woods for very quickly sending me a copy of his thesis. Thank you Ingrid van Loon for putting up with my persistent deadline pushing procrastination and general tendency towards disorganization.

It is also a great joy to thank all the wonderful friends I have met in Amsterdam, and my parents for letting me be so far away from them!

Contents

Chapter 1

Introduction

1.1 Definability over the Natural Numbers

Decidability questions have provided a large impetus to the study of definability of arithmetical predicates over the natural numbers. As elementary number theory $\langle \omega, +, \cdot \rangle$ is undecidable, the theory $\langle \omega, \mathcal{N} \rangle$, for a set of predicates $\mathcal{N}$, is undecidable if addition and multiplication can be interpreted within this theory—that is, roughly speaking, if there is a first-order sentence defining addition and multiplication from predicates in $\mathcal{N}$. Results along these lines include:

- addition and the predicate $\mathrm{SQ}(x, y)$, which holds iff $x^2 = y$, can first-order define multiplication [TMR53]

- successor and multiplication can first-order define addition [Rob49]

- successor and the predicate $\mathrm{DIVIDES}(x, y)$, which holds iff x divides y, can first order define multiplication [Rob49]

- the ordering relation $<$ and the predicate $\mathrm{RELPRIME}(x, y)$, which holds iff the greatest common divisor of x and y is one, can first-order define multiplication [Woo81]

On the other hand, Presburger [Pre29] showed that the theory $\langle \omega, + \rangle$ is decidable. The transition between decidability and undecidability is still not well understood—that is, to our knowledge, no recursive predicate $P(x)$ is known for which the theory $\langle \omega, +, P \rangle$ is undecidable yet unable to define *all* recursive predicates.

1.2 Definability over Finite Structures

A major motivation to the study of definability over finite structures is the intimate connection between logical expressiveness and complexity. This connection was first brought out by Fagin [Fag74], who showed that the sets of structures in **NP** are exactly those which can be defined by existential second-order sentences. Since this initial result, the field of descriptive complexity has developed logical equivalents for most complexity classes, including **P** [Imm82, Var82], turning the problem of showing $\mathbf{P} \neq \mathbf{NP}$ into a problem of logical definability.

When translated into logic, a separation result between complexity classes becomes an inexpressibility result. That is, suppose A, B are complexity classes equivalent to the logics $\mathcal{L}_A, \mathcal{L}_B$, respectively. Then if A is strictly contained in B, there must be a language in B which cannot be defined by $\mathcal{L}_A$. Thus in logical terms a lower bound result shows that a certain amount of *expressiveness* is required to define a problem, rather than a certain amount of time or space. The main tool to prove inexpressibility results is Ehrenfeucht-Fraïssé games, and with the equivalence between expressiveness and complexity, this becomes another tool to separate complexity classes.

1.3 Decidability and Counting

As with the gap between decidability and undecidability, we also see a gap in definability on finite structures. Considering predicates on finite structures, a useful concept is how high the set of predicates can 'count'—roughly speaking, in a string of zeros and ones of length n, how many ones a first-order formula over this set of predicates can accurately identify. It is known that PLUS cannot count beyond a constant, and that PLUS and TIMES can count polylogarithmically in n. But no natural set of predicates is known which can count beyond a constant but which is without the full definability of PLUS and TIMES. In this thesis, we rule out two more hopeful sets of predicates for this property, $(<, \text{TIMES})$ and $(<, \text{DIVIDES})$.

1.4 Outline

Chapters 2 and 3 introduce the logical and complexity background needed for the rest of the thesis. Chapter 4 gives an introduction to Ehrenfeucht-Fraïssé games and tries to develop a generalized game technique for separating the expressiveness of logics and therefore separate complexity classes. In Chapter 5, we discuss what is know about the structural complexity of addition, multiplication, and division, and the advantages of the descriptive approach to the complexity of these problems. In Chapter 6 we show that Julia Robinson's [Rob49] two main results on arithmetic definability over the natural numbers – that multiplication and ordering can define addition, and that the divisibility relation and ordering can define multiplication – can both be extended to finite structures [1]. The proof that multiplication and ordering can first-order define addition proceeds by first showing that multiplication and ordering can first-order define the BIT predicate, improving the previous result $\mathrm{FO}(<,\mathrm{BIT}) = \mathrm{FO}(<,\mathrm{PLUS},\mathrm{TIMES})$ to $\mathrm{FO}(<,\mathrm{BIT}) = \mathrm{FO}(<,\mathrm{TIMES})$. We then use a standard construction of addition with the BIT predicate to show that multiplication and ordering can first-order define addition. In conjunction with our result that $\mathrm{FO}(<,\mathrm{BIT}) = \mathrm{FO}(<,\mathrm{TIMES})$, a recent result regarding the Crane Beach Conjecture can be used to show that addition and ordering cannot first-order define multiplication. This is the basis of our claim that multiplication is indeed harder than addition.

[1] Actually what Julia Robinson shows is that multiplication and *successor* can define addition and that divisibility and *successor* can define multiplication. These results are stronger than results using ordering, as ordering can define successor, but successor cannot define ordering. We do not consider definability with successor in this thesis, as ordering is a much more natural relation on finite structures. On the other hand, Julia Robinson was interested in applying her results to axiomatizations of arithmetic and decidability, where successor is a more natural primitive.

Chapter 2

Background in Logic

2.1 Preliminary Definitions

A *vocabulary* $\tau = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$ consists of relation symbols and constant symbols. The relation symbol $R_i^{a_i}$ is of arity a_i. An interpretation to these relational and constant symbols is given in a *structure.* A structure $\mathcal{A}$ with vocabulary τ is a tuple

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

The *universe* of $\mathcal{A}$ is the nonempty set $|\mathcal{A}|$. The relation $R_i^{\mathcal{A}}$ is an interpretation in the universe $|\mathcal{A}|$ of the relation symbol $R_i^{a_i}$. That is, $R_i^{\mathcal{A}} \subseteq |\mathcal{A}|^{a_i}$. And for each constant symbol c_j there is a specified element $c_j^{\mathcal{A}} \in |\mathcal{A}|$.

It is a basic tenet of descriptive complexity to work over finite universes, as computers necessarily deal with finite objects. Thus in our case generally $|\mathcal{A}| = \{0, \ldots, n-1\}$.

The main structures we will be looking at in this thesis are structures of binary strings, with vocabulary $\tau_s = \langle <^2, I_1^1, 0, 1 \rangle$, and universe $\mathcal{S} = \{0, \ldots, n-1\}$. The elements of $\mathcal{S}$ are thought of as positions in the string, $<$ is the usual ordering on these positions, and the set I_1 represents a particular string s under the interpretation that a 1 is written at position k of s iff $k \in I_1$. We also assume that the universe $\mathcal{S}$ contains at least two nonequal elements, denoted by the constants 0 and 1.

2.2 Definability

Structures are the objects which logical languages speak about. Exactly which properties we can express about a structure depends on the logical language we are using. In this thesis, we will be mainly considering the expressiveness of first-order languages with arithmetic predicates.

The first-order language for a vocabulary τ, denoted as FO_τ, consists of the set of formulas built up from the relation and constant symbols of τ, the logical relation $=$, logical connectives $\wedge, \vee, \neg$, variables $x, y, z, \ldots$, and quantifiers $\forall, \exists$. When the vocabulary is unambiguous, we will drop the subscript τ.

To do arithmetic, we will be supplementing the first-order language with additional predicates:

- $\mathrm{PLUS}(i, j, k)$ which holds iff $i + j = k$ for positions i, j, k
- $\mathrm{TIMES}(i, j, k)$ iff $i \cdot j = k$
- $\mathrm{DIVIDES}(i, k)$ iff there is a j such that $i \cdot j = k$
- $\mathrm{BIT}(i, j)$ iff bit j of the binary representation of i is 1.

For a set of numerical predicates $\mathcal{N}$, we denote by $\mathrm{FO}(\mathcal{N})$ the first-order language where formulas are also allowed to include predicates from $\mathcal{N}$. We say that a set of numerical predicates $\mathcal{N}$ can first-order define a k-ary predicate P iff there exists a $\mathrm{FO}(\mathcal{N})$ sentence ϕ such that for all $x_1, \ldots, x_k$,

$$P(x_1, \ldots, x_k) \iff \phi(x_1, \ldots, x_k)$$

As an easy example, note that the predicate PLUS can define $<$ as follows:

$$x < z \iff \exists y(\mathrm{PLUS}(x, y, z) \wedge \neg\mathrm{PLUS}(y, y, y))$$

Thus $\mathrm{FO}(<, \mathrm{PLUS})$ and $\mathrm{FO}(\mathrm{PLUS})$ have the same expressive power, and in general we use this fact to our notational convenience by not explicitly listing $<$ in a set of predicates that contains PLUS.

2.3 How Logic Defines a Problem

To illustrate the connection between complexity classes and logical languages, we first more formally define what a problem is. A *problem*, or alternatively a *property*, is a set of structures of a given vocabulary. For example, in the vocabulary of graphs, the problem of three-colorability is the set of graphs which are three colorable. We can also think of this set as representing the property of being three-colorable. When in the vocabulary of strings, we will also refer to a problem as a language, because in this case a set of structures is a set of words on the alphabet {0,1}.

Now the ability of a logical formula to define a problem becomes clear, because a logical formula also specifies a set of structures, namely the set of structures which are models of that formula. Continuing the above example, the problem three-colorability is definable in a logic $\mathcal{L}$ iff there exists a formula $\phi \in \mathcal{L}$ such that the graphs satisfying ϕ are exactly the 3-colorable graphs.

This way of thinking can be extended to see the correspondence between a logic and a complexity class.

We say that a logic $\mathcal{L}$ *captures* a complexity class $\mathcal{C}$ iff the following hold:

- for any $\mathcal{C}$-machine T, there is a formula ϕ of $\mathcal{L}$ whose models are exactly those structures accepted by T.
- for any formula ϕ of $\mathcal{L}$, there is a $\mathcal{C}$-machine T which accepts exactly those structures that are models of ϕ.

In the next chapter we give a concrete example of this correspondence, showing that first-order logic with arbitrary numerical predicates captures the circuit complexity class AC^0.

Chapter 3

Circuit Models

3.1 Why Circuit Models?

The arithmetic operations we will be considering in this thesis are all relatively simple in terms of complexity — in fact they can all be computed in logarithmic space. To achieve greater resolution in classifying the complexity of arithmetic, we must look at the smaller complexity classes described by circuit models. As circuit models are perhaps less familiar than Turing machine models, we will give a short overview.

Circuit models are roughly based on the design of circuits in computers. These abstracted circuits are represented as directed acyclic graphs, with nodes labelled as input, constant, AND, OR, NOT, or output nodes. Input and constant nodes have zero fan-in, while output nodes have fan-out zero. We will consider circuits with a single output node, thus circuits which compute a boolean function $f : \{0,1\}^n \to \{0,1\}$. The relevant resource measures of a circuit are its size, that is its number of nodes, and its depth, the length of the longest path in the circuit.

A salient difference between the circuit model and the Turing machine model is that a circuit has a fixed input size n, its number of input nodes, while a Turing machine can accept variably sized input on its tape. Thus to do real computation, we need a family of circuits $C = \{C_i\}_{i \in N}$. We say that a circuit family C accepts the language L iff for all $n \in N$ and $w \in \{0,1\}^n$

$$w \in L \iff C_n \text{ outputs 1 on input } w$$

3.2 Uniformity

One subtlety of circuits absent from Turing machine models is uniformity. Uniformity imposes restrictions on how complicated a circuit can be to *build.* A circuit family C is $\mathcal{C}$ uniform iff there is a $\mathcal{C}$ Turing machine which outputs a description of the circuit C_n on input 0^n. A non-uniform circuit family is acutally able to do uncomputable things, as a constant gate could be described as being 1 if the nth Turing machine halts and 0 otherwise. In a $\mathcal{C}$ uniform circuit family, on the other hand, the value of constant gates are computable in $\mathcal{C}$. Thus uniformity restrictions limit the amount of precomputation that can be built into the design of the circuit.

In choosing a uniformity class, we want something that is computationally weaker than the circuit class itself—therefore no precomputation can be built into the circuit which could not theoretically be done by the circuit itself. On the other hand, we still want a uniformity condition which gives a circuit class that is robust under small changes in the input and encoding.

The most widely accepted uniformity condition for circuit classes is DLOGTIME uniformity, equivalent to $\mathrm{FO}(<, \mathrm{BIT})$ uniformity [BIS90].

3.3 AC^0 and TC^0

Two circuit classes relevant to arithmetical operations are AC^0 and TC^0. The class AC^i consists of languages accepted by circuits which are of polynomial size and of depth $O(\log^i(n))$. Thus AC^0 circuits are of polynomial size and constant depth. In AC^i circuits, nodes are labelled with AND, OR, NOT, and constant gates, where the AND and OR gates are allowed to be of unbounded fan-in.

The circuit class TC^i generalizes AND and OR gates to allow threshold gates. A threshold gate with threshold k evaluates to 1 iff the sum of its inputs is greater than or equal to k. Again TC^i circuits are polynomially sized and of depth $O(\log^i(n))$. It is easy to see $\mathrm{AC}^0 \subseteq \mathrm{TC}^0$ as an AND gate is a threshold gate with the threshold set to the number of inputs, and an OR gate is a threshold gate with a threshold of one.

3.4 Correspondence between FO and AC^0

The equivalence between $\mathrm{FO}(<, \mathrm{PLUS}, \mathrm{TIMES})$ and DLOGTIME uniform AC^0 was mentioned in the introduction. In this and the next section, we will sketch a proof of this result. We start with the easier task of showing equivalence in the nonuniform case.

Theorem 3.4.1 ([Imm87]) *Let L be a language over the alphabet $\{0, 1\}$. Then the following are equivalent:*

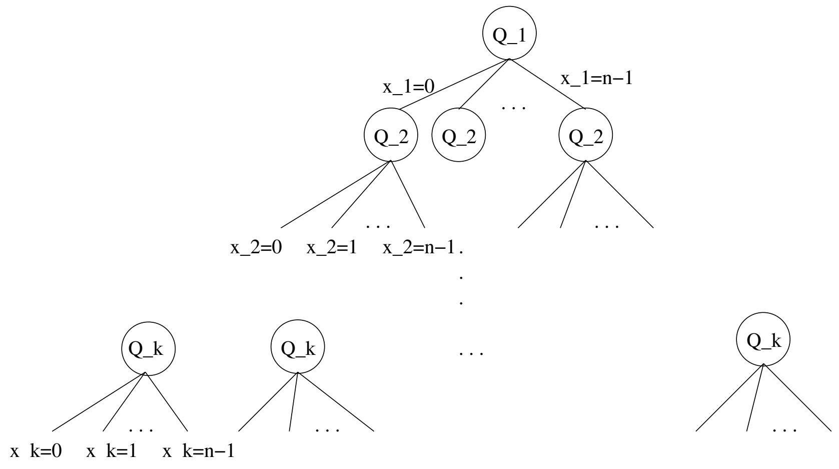
1. *L is recognized by a family of (possibly nonuniform) AC^0 circuits*
2. *There is a set of numerical predicates $\mathcal{N}$ and a sentence $\phi \in$ FO($\mathcal{N}$) such that L is the set of models of ϕ.*

Proof ($2 \Rightarrow 1$) We may assume ϕ is in prenex form and of quantifier depth k. Thus we can write ϕ as $Q_1x_1 \ldots Q_k\sigma$ where each Q_i is either a $\forall$ or a $\exists$ quantifier, and σ is quantifier free. Fix the input size n. We will build an AC^0 circuit which accepts all the strings of length n which are models of ϕ.

Within the circuit, we represent a $\forall$ quantifier as an AND gate with n inputs. Thus for the output of the gate to be true, the input from each position must be true.

A $\exists$ quantifier is represented within the circuit by an OR gate with n inputs. Thus the output of this gate is true if the input from at least one position is true.

We construct a tree of gates following the quantifiers in ϕ, as depicted in the figure below. The size of this tree is n^k, and on the n^k paths from the root to a leaf, all possible assignments of n positions to the k variables are realized.



What remains is to evaluate σ at each leaf. As σ is quantifier free, it is a boolean combination of of (possibly negated) atomic expressions—expressions of the form $x_i = x_j, I(x_i)$ or $P(t_1, \ldots, t_m)$ where $P \in \mathcal{N}$ is an m-ary predicate, and each $t_i \in \{x_1, \ldots, x_k\}$. At each leaf, the variables $x_1, \ldots, x_k$ have assumed a definite numerical value. This in turn determines a numerical value 0 or 1 for each the atomic expressions mentioned above. Thus We can represent the evaluation of these expressions with a constant 0 or 1 gate, and combine these gates according to σ using the AND, OR, and NOT gates at our disposal. The number of symbols in σ does not depend on n, thus the part of the circuit representing σ will have a constant number of gates.

($1 \Rightarrow 2$): This inclusion is more difficult, and we only sketch the idea here. What we wish to illustrate is how the computation of a circuit of fixed size C_n can be simulated in a first-order language by adding new numerical predicates.

As C_n is an AC^0 circuit, it must be of polynomial size and so have fewer than n^k nodes for some k. Thus we can number each node by using a vector $\mathbf{x}$ of k variables. We can then define numerical predicates describing the gate type at each node number:

- $INP(\mathbf{x}, t)$ meaning the gate labelled by vector $\mathbf{x}$ is an input gate, inputting position t of the input string
- $OUT(\mathbf{x})$ the gate labelled by $\mathbf{x}$ is the output gate
- $AND(\mathbf{x}), OR(\mathbf{x}), \text{and } NOT(\mathbf{x})$ to denote the type of gate at $\mathbf{x}$
- $CHILD(\mathbf{x}, \mathbf{y})$ to mean that gate $\mathbf{x}$ is a child of gate $\mathbf{y}$

Now we can start to model the flow of the computation through the circuit by defining more predicates such as "$\mathbf{x}$ is an input gate and evaluates to one on the given input":

- $INP(\mathbf{x}, t) \wedge I_1(t) \rightarrow ONE(\mathbf{x})$

Then we can describe when the gates one level up from the input evaluate to one, for example to express "gate x is an AND gate at level 1 which evaluates to 1" is the conjunction of the following formulas:

- $AND(\mathbf{x})$
- $\forall \mathbf{y}(CHILD(\mathbf{y}, \mathbf{x}) \rightarrow \exists t\ INP(\mathbf{y}, t))$
- $\forall \mathbf{y}(CHILD(\mathbf{y}, \mathbf{x}) \rightarrow ONE(\mathbf{y}))$

We can continue in this manner defining "gate $\mathbf{x}$ is a gate at level d with value 1," and so on until we reach "gate $\mathbf{x}$ is an output gate with value 1". Then the circuit accepts a string iff this predicate is true.

3.5 FO($<$, BIT) = DLOGTIME AC^0

The above argument, with more attention to uniformity, yields the following theorem:

Theorem 3.5.1 *Let L be a language over the alphabet $\{0,1\}$. Then the following are equivalent:*

1. *L is recognized by a DLOGTIME AC^0 family of circuits*
2. *There is a sentence $\phi \in \mathrm{FO}(<, \mathrm{BIT})$ whose set of models is L*

Proof ($1 \Rightarrow 2$): We follow the previous proof, additionally armed with the fact that DLOGTIME $\subseteq$ FO($<$, BIT), Prop 8.1 of [BIS90]. This means that if the description of the circuit—the parent-child relationships and gate type predicates—can be computed in DLOGTIME, then they can also be defined in FO($<$, BIT).

($2 \Rightarrow 1$): This direction hinges on the fact that the BIT predicate can be simulated in DLOGTIME, Lemma 7.1 of [BIS90]. The only part of the construction in the previous thm that was potentially nonuniform was the evaluation of the atomic expressions in the quantifier free part of ϕ. In FO($<$, BIT), the atomic expressions are of the form $x_i \leq x_j, I_1(x_i)$, and $\mathrm{BIT}(x_i, x_j)$. All of these can be computed in DLOGTIME. ▮

Chapter 4

Ehrenfeucht-Fraïssé games

4.1 How the game is played

Ehrenfeucht-Fraïssé games are the main tool for proving inexpressibility results in first-order logic. The game is played on two structures, $\mathcal{A}$ and $\mathcal{B}$, of the same vocabulary τ. There are two players in the game, Spoiler and Duplicator. Spoiler, with his moves, is trying to point out a difference between the structures, while Duplicator chooses her moves to imitate Spoiler and mask any differences between the structures.

Spoiler moves first by picking an element in one of the structures. Duplicator then moves, choosing an element in the other structure. Play continues in this way, at each turn Spoiler being at liberty to choose in which structure he plays. At the end of k rounds, elements $a_1, \ldots, a_k$ have been chosen from $\mathcal{A}$, and elements $b_1, \ldots, b_k$ have been chosen from $\mathcal{B}$. Duplicator wins k-round Ehrenfeucht-Fraïssé game for FO($\mathcal{N}$) iff the mapping $a_i \mapsto b_i$ is a partial $\tau \cup \mathcal{N}$ isomorphism. That is, iff

1. for all $i, j \leq k, a_i = a_j \iff b_i = b_j$
2. for every m-ary relation symbol R in τ, and all $i_1, \ldots, i_m \leq k$, it holds that $R^{\mathcal{A}}(a_{i_1}, \ldots, a_{i_m}) \iff R^{\mathcal{B}}(b_{i_1}, \ldots, b_{i_m})$
3. for every m-ary predicate $\mathcal{P} \in \mathcal{N}$, and all $i_1, \ldots, i_m \leq k$, it holds that $\mathcal{P}(a_{i_1}, \ldots, a_{i_m}) \iff \mathcal{P}(b_{i_1}, \ldots, b_{i_m})$

Duplicator has a winning strategy iff she can win the game for any choice of spoiler's moves. As the game is determined, Spoiler has a winning strategy iff Duplicator does not.

The usefulness of the game comes from the following theorem. Recall that a property $\mathcal{P}$ is a set of structures of a given vocabulary.

Theorem 4.1.1 *(Ehrenfeucht, Fraïssé) A property $\mathcal{P}$ is definable in* $\mathrm{FO}(\mathcal{N})$ *iff there is a $k \in N$ such that for every $\mathcal{A} \in \mathcal{P}, \mathcal{B} \notin \mathcal{P}$, Spoiler has a winning strategy in the k-round game for* $\mathrm{FO}(\mathcal{N})$ *on $\mathcal{A}$ and $\mathcal{B}$.*

We can transform this theorem into a necessary and sufficient condition for the inexpressibility of a property over $\mathrm{FO}(\mathcal{N})$ as follows. If for some property $\mathcal{P}$ and $\mathcal{A} \in \mathcal{P}$ and $\mathcal{B} \in \bar{\mathcal{P}}$ Duplicator has a winning strategy in the Ehrenfeucht-Fraïssé game for $\mathrm{FO}(\mathcal{N})$ of length k, then $\mathcal{P}$ cannot be expressed by a $\mathrm{FO}(\mathcal{N})$ sentence of quantifier depth k. For if it could, then by definition there would be a sentence ϕ of quantifier depth k which is satisfied by $\mathcal{A}$ but not by $\mathcal{B}$. Then by theorem 4.1.1, Duplicator would not have a winning strategy, a contradiction.

Theorem 4.1.2 *A property $\mathcal{P}$ is not definable in* $\mathrm{FO}(\mathcal{N})$ *iff for all $k \in N$ there exists $\mathcal{A} \in \mathcal{P}, \mathcal{B} \notin \mathcal{P}$ such that Duplicator has a winning strategy in the k-round game for* $\mathrm{FO}(\mathcal{N})$

We will show how these theorems can be used with a simple example. Over the vocabulary of strings consider the property or language $L = 0^*1^*$, the set of strings which can be written as the concatenation of a string consisting only of zeros followed by a string consisting only ones. This language can be defined in FO(<) with the following sentence.

$$\exists x \forall y (y \leq x \rightarrow \neg I_1(y) \wedge x < y \rightarrow I_1(x)$$

This sentence asserts the existence of a position x, before which occur only zeros and after which occur only ones. Now we wish to show that this language cannot be defined in FO when ordering is not present, using theorem (4.1.2).

Fix $k \in N$. Let $u = 000111 \in L$ and $v = 111000$. We define a strategy for Duplicator to win the k-round game for FO on these strings. If Spoiler plays at position a_i on u, then Duplicator chooses $b_i = n - a_i$ to play in v. Similarly, if Spoiler instead plays into v with b_i, then Duplicator moves to $a_i = n - b_i$. Note that if $a_i = n - b_i$ then $n - a_i = b_i$, thus to show that this strategy is a win for duplicator, we have to show that the mapping $a_i \mapsto b_i = n - a_i$ is a partial isomorphism. Looking at the conditions listed above we have to check:

1. $a_i = a_j \iff n - a_i = n - a_j$
2. $I_1^u(a_i) \iff I_1^v(n - a_i)$ as v is the string u written backwards

3. As we have no numerical predicates, there is nothing more to check!

Now we have shown that there is a language, namely 0^*1^*, which is expressible in FO($<$), but not in FO. This is a simple example of how Ehrenfeucht-Fraïssé games can be used to separate logics, which, coupled with the equivalence of logics and complexity classes, can be used to separate complexity classes.

There are limits to the capability of this technique in separating larger complexity classes, but Ehrenfeucht-Fraïssé type games have, for example, shown that monadic **NP** is different from monadic **coNP** [Fag75]. To extend the applicability of Ehrenfeucht-Fraïssé type games we next talk about developing a more general technique for Ehrenfeucht-Fraïssé games to prove inexpressibility results.

4.2 Generalized Games for Inexpressibility

We wish to formulate a general game technique to prove inexpressibility results. Ehrenfeucht-Fraïssé games can be used to prove inexpressibility results for FO($\mathcal{N}$) because the conditions for a winning strategy completely characterizes expressibility in FO($\mathcal{N}$). To use this technique towards separating more complexity classes, we must first develop games to characterize, in the sense of thm (4.1.1), other logics relevant to complexity. For logics more expressive than FO, characterizing games will have more stages and components than the standard Ehrenfeucht-Fraïssé game to represent this added expressitivity. For example, the game developed by Ajtai and Fagin [AF90] characterizing monadic existential second-order logic (termed monadic **NP**) begins with a round of "coloring" the structures. Coloring is really the selection of sets, the set of elements that are red, the set of elements that are green, ... and represents within the game the quantification over sets possible with existential second-order quantifiers in monadic **NP**.

Say that we wish to prove inexpressibility results about a class of sentences $\mathcal{C}$. The first step is to design a game with parameters P (including factors like number of rounds, number of colors, etc.), called the Name Game after ourselves, which characterizes $\mathcal{C}$ in the sense that we can prove the theorem

Theorem 4.2.1 *Duplicator has a winning strategy in the parameter P Name Game on structures $\mathcal{A}$ and $\mathcal{B}$ iff $\mathcal{A}$ and $\mathcal{B}$ agree on the sentences of $\mathcal{C}$.*

Then we will also get the companion theorem

Theorem 4.2.2 *$\mathcal{P}$ is not expressible in the class of sentences $\mathcal{C}$ iff for every set of parameters P there is $\mathcal{A} \in \mathcal{P}$ and $\mathcal{B} \in \bar{\mathcal{P}}$ such that duplicator has a winning strategy in the P Name Game on $\mathcal{A}$ and $\mathcal{B}$.*

Ajtai and Fagin used this game technique to show that graph connectivity is not expressible in monadic **NP**, thus separating monadic **NP** from monadic **coNP**.

Chapter 5

With All and One, Can You Say Some?

A general theme of this thesis is how logical techniques can help to answer problems in complexity. In this chapter we pause to remember that the relationship is mutual, with an example of how results from complexity can help to answer logical questions.

The familiar quantifiers $\forall$ and $\exists$ can be used to express that all members or at least one member of a finite structure has a given property. Our question is the following: Is there some first-order formula which can express that *most* members of a finite structure have a given property?

5.1 TC^0 and Majority

Results from complexity can help us answer this question. In chapter 3 we introduced the circuit class TC^0 consisting of polynomially sized, constant depth circuits with threshold, NOT, and constant gates. We now show that TC^0 can be equivalently defined with majority gates instead of threshold gates. The output of a majority gate is 1 iff a majority of its inputs are 1.

Consider a threshold gate with threshold k and s inputs. To construct a majority gate with the same behavior, we can pad the input with t constant 0 or 1 gates such that exactly k inputs must be 1 in order to have a majority. That is, we choose t such that $\lceil \frac{s+t}{2} \rceil = k$.

- $2k - s > 0$ means that k is larger than half the inputs, thus we add t more inputs with 0.
- $2k - s = 0$ in this case we do not have to do anything.

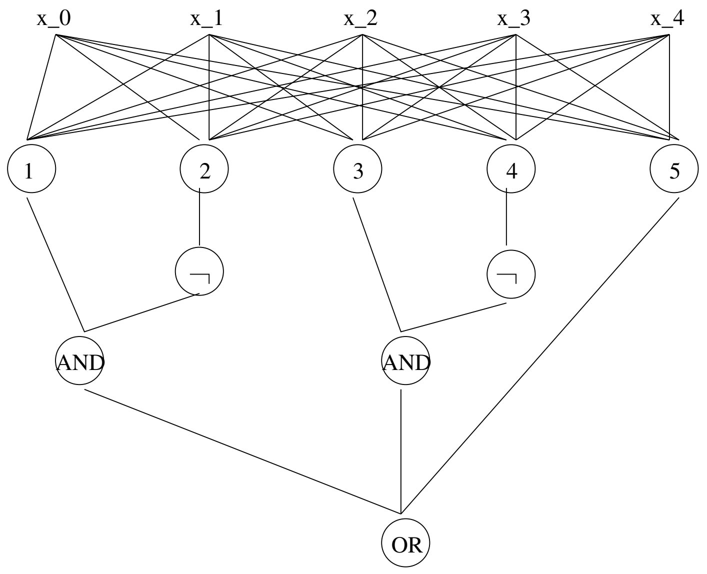
- $2k - s < 0$ means that k is less than half the inputs, thus we pad with t inputs of 1.

If we replace every threshold gate with a majority gate in this way, our new circuit will still be polynomially sized as we add at most n constant nodes to each of n^l nodes in the circuit, thus we add at most n^{l+1} nodes. The depth of the circuit does not change. Thus every circuit in TC^0 can be transformed into a polynomially sized, constant depth circuit with majority, NOT, and constant gates.

Another thing to consider is the uniformity of our circuit. From a plan of the original circuit, to construct the new circuit the only calculations we have to make is $2k - s$. We know that this calculation can be defined in FO(BIT) or equivalently in DLOGTIME. Thus for any TC^0 circuit with uniformity DLOGTIME or greater, we can construct a majority circuit of the same uniformity.

Now we can follow the same argument used in thm (3.5.1) to show that FO(BIT) coincides with DLOGTIME uniform AC^0 to show that FOM coincides with DLOGTIME uniform TC^0. Thus our original question can now be rephrased as does $\mathrm{AC}^0 = \mathrm{TC}^0$?

These complexity classes have been separated. The language PARITY, containing the strings with an odd number of ones can be decided by a very uniform TC^0 circuit. An example of the circuit for $n = 5$ is given in the figure below. Numbers in the circles indicate thresholds.



In general the design of the circuit is to solve the problem as follows. Assume the input length $n = 2\ell - 1$ is odd. A $2\ell - 1$-bit string has an odd number of ones iff it has exactly $2k - 1$ ones for some $1 \leq k \leq \ell$. This string has exactly $2k - 1$ ones iff the gate with threshold $2k$ fails and the one with $2k - 1$ passes. By ORing together circuits to test for $2k - 1$ ones for $1 \leq \ell \leq k$, we obtain our threshold circuit for parity.

On the other hand, parity cannot be decided by DLOGTIME uniform AC^0 circuits [FSS84]. This is a very interesting lower bound result, but is outside the scope of this thesis. This result shows $\mathrm{AC}^0 \subset \mathrm{TC}^0$, and thus there is no first-order formula which can define the majority quantifier.

To put this more figuratively, with 'all' and 'one', we cannot say 'most', or even k for a constant k. Unfortunately this discussion does not help to separate the meaning of the words 'couple', 'few', and 'some'.

Chapter 6

Structural Complexity

A difficulty in classifying the complexity of arithmetical operations is that, as these problems are relatively simple computationally, their complexity is sensitive to changes in representation. In descriptive complexity, problems and inputs are represented as predicates, and thus the particularities of how inputs and outputs are encoded can be avoided. This leads to a more robust classification of the complexity of arithmetical problems. In this chapter we survey the structural complexity of arithmetical problems before turning to their descriptive complexity in the next.

6.1 The long and short of it

The short answer is Logspace. Addition, subtraction, multiplication, and division of n-bit numbers can all be done in space logarithmic in n. Division was only shown to be in Logspace relatively recently [CDL].

For a longer answer, we must pay attention to how each problem is encoded, and if we are working with long ($O(n)$ bits) or short ($O(\log n)$ bits) numbers. Logically, short numbers can be represented directly as variables, or vectors of variables, whereas long numbers must be represented with predicates. For example, the BIT predicate allows us to make reference to numbers as large as $2^n - 1$ when the size of our structure is only n.

The circuit and descriptive complexity classes considered in this thesis are closed under a polynomial change in the input size. The easiest way to see this is that a polynomial change in the input size will only cause the size of our circuit to change by a polynomial, and thus the circuit will still be in the same class. That the descriptive complexity classes we consider are also closed under a polynomial change in input size can be seen by their equivalence with the respective circuit classes. For notational convenience in the following, we will usually specify an input size, but keep in mind that a result for input size n or $\log n$ can be generalized to long or short numbers.

6.1.1 Addition

Consider first the problem of adding two n-bit numbers, $a = a_{n-1} \dots a_0$ and $b = b_{n-1} \dots b_0$. When the input of a and b is given in the form $a_0 b_0 a_1 b_1 \dots a_{n-1} b_{n-1}$, then a constant space Turing machine can output their sum. If, on the other hand, the input is given sequentially $a_{n-1} \dots a_0$ $b_{n-1} \dots b_0$, then the problem can no longer be solved in a constant amount of space [1].

Using predicates we can consider both the decision and graph versions of the addition problem on n-bit numbers:

- $\mathrm{ADD}(X, Y, i)$ which holds iff the ith bit of $X + Y$ is 1, and
- $\mathrm{ADD}(X, Y, Z)$ which holds iff $X + Y = Z$

These two problems are $\mathrm{FO}(<, \mathrm{BIT})$ reducible to each other, and both can be solved by DLOGTIME uniform AC^0 circuits of size $O(n^3)$. To see that these problems are in DLOGTIME uniform AC^0, we give a $\mathrm{FO}(<, \mathrm{BIT})$ formula expressing $\mathrm{ADD}(X, Y, i)$.

We use the "carry-look-ahead" algorithm. To compute the ith bit of $X + Y$ we first see if a carry has been propagated to bit i:

$$\begin{aligned} \phi_{carry}(X, Y, i) &\equiv \exists j (j < i \wedge \mathrm{BIT}(X, j) \wedge \mathrm{BIT}(Y, j) \\ &\wedge \forall k (j < k < i \rightarrow BIT(X, k) \vee \mathrm{BIT}(Y, k)) \end{aligned}$$

The formula $\phi_{carry}(X, Y, i)$ holds if a carry is generated at a position less than i and is then propagated through all the intervening positions. Now the ith bit of $X + Y$ will be 1 if exactly one of $\mathrm{BIT}(X, i), \mathrm{BIT}(Y, i), \phi_{carry}(i)$ hold, or if all three of them hold. The easiest way to express this is with the exclusive or operation $\oplus$,

$$\alpha \oplus \beta \equiv \alpha \leftrightarrow \neg \beta \tag{6.1}$$

$$\phi_{add}(X, Y, i) \equiv (\mathrm{BIT}(X, i) \oplus \mathrm{BIT}(Y, i)) \oplus \phi_{carry}(X, Y, i) \tag{6.2}$$

[1]For a proof, note that the language $a_{n-1} \dots a_0 e b_{m-1} \dots b_0 e c_m \dots c_0$, where $a_{n-1} \dots a_0 + b_{m-1} \dots b_0 = c_m \dots c_0$ in binary and $m \geq n$, has infinitely many Myhill-Nerode equivalence classes. Thus by the Myhill-Nerode Thm, there is no deterministic finite automaton which accepts this language. On the other hand, the language with the bits interleaved $a_0 b_0 c_0 \dots a_m b_m c_m$ has, by our count, 16 Myhill-Nerode equivalence classes.

	0	…	0	x_{n-1}	0	…	0	x_{n-2}	…	…	0	…	0	x_0
$\times$	0	…	0	1	0	…	0	1	…	…	0	…	0	1
	0	…	0	x_{n-1}	0	…	0	x_{n-2}	…	…				
	0	…	0	x_{n-2}	0	…	0	x_{n-3}	…	…				
				$\vdots$					…	…				
	0	…	0	x_0	0	…	0	0	…	…				
				P					…	…				

Table 6.1: Illustration of a first-order reduction from PARITY to MULT

6.1.2 Multiplication

We use the predicate MULT(X, Y, i) to represent multiplication of n-bit numbers. Again there is a FO$(<, \mathrm{BIT})$ equivalence between this predicate and the multiplication graph predicate MULT(X, Y, Z) [2].

Multiplication of n-bit numbers cannot be done by constant depth, polynomially sized circuits, as there is a FO$(<, \mathrm{BIT})$ reduction from PARITY to MULT(X, Y, i) [FSS84].

The idea of the reduction is best illustrated by Table (6.1). To determine the parity of $X = x_{n-1} \ldots x_0$ we construct two n^2 bit numbers, $\bar{X}, \bar{Y}$ where

$$\begin{aligned} \bar{X} &= \sum_{i=0}^{n-1} x_i 2^{in} \\ \bar{Y} &= \sum_{i=0}^{n-1} 2^{in} \end{aligned}$$

In the product

$$\bar{X}\bar{Y} = \sum_{i=0}^{n-1} c_i 2^{in}$$

each c_i is a sum of at most n bits, and thus its binary representation will fit in $\log n$ so that there are no carries from one block to the next. Then the low order bit of c_{n-1}

$$c_{n-1} = \sum_{i=0}^{n-1} x_i$$

[2]From a descriptive complexity point of view, this equivalence is easy to see. Thinking solely in terms of AC^0 reductions, however, it is not as obvious and Sam Buss wrote a small note [Bus91] which goes through the details. Buss was interested in this question in part because the non-existence of constant depth polynomial size circuits for the graph of multiplication implies that the Δ_0^b-predicates of bounded arithmetic are not in AC^0.

holds the parity of the sum of the x_i's.

As with parity, multiplication of long numbers can be computed with TC^0 circuits. In fact, unbounded fan-in, constant depth circuits with NOT gates, AND gates, and gates for multiplication exactly capture TC^0 [Bus91]. To our knowledge it is still an open problem, conjectured to be false, if multiplication can be reduced to parity.

6.1.3 Division

Out of the familiar arithmetical operations, the complexity of division was the most difficult to classify. In fact it was not until recently that division found its complexity "home", a long process finished by Bill Hesse in showing that the predicate DIVISION(X, Y, i) is complete for DLOGTIME TC^0 [Hes01].

6.2 Preview

In the next chapter, we will be examining the complexity of addition and multiplication of short numbers. These problems will be represented respectively by the predicates PLUS(i, j, k) and TIMES(i, j, k), where i, j, k are variables thought of as representing positions in a string. Both of these predicates can be expressed in FO$(<, \text{BIT})$, and so are in DLOGTIME uniform AC^0. What we will show in the next chapter is that, although it is doubtful we can separate these problems in terms of circuit complexity classes, we can separate them in terms of descriptive complexity classes — in fact FO$(<, \text{PLUS}) \subset$ FO$(<, \text{TIMES})$.

Chapter 7

Arithmetical Definability

Julia Robinson's paper [Rob49] contains two main results about first-order arithmetical definability over the natural numbers. First, that multiplication and successor can define addition, and secondly that the divisibility relation and successor can define multiplication, and hence also addition.

In this chapter we show that these results can be transferred to the finite case, where we use the ordering relation instead of successor.

7.1 FO(<, BIT) = FO(<, TIMES)

Robinson defined addition with multiplication and successor with the following formula:

$$x + y = z \iff S(x \cdot z) \cdot S(y \cdot z) = S(z \cdot z \cdot S(x \cdot y)).$$

This formula, however, cannot be naïvely applied to define addition over finite structures. If our inputs are of order k, then to check this equality, we will have to store a number of order k^4, exceeding the size of our structure when $k > n^{(1/4)}$.

We take a different approach to defining addition in terms of multiplication and ordering. First we show the BIT predicate can be defined in terms of multiplication and ordering. We do this by following the existing proof that FO($<$, BIT) = FO(PLUS, TIMES) [Imm88], modifying the second half of this proof to show that the use of PLUS is not essential. Then (6.2) gives a definition of addition in terms of BIT.

Theorem 7.1.1 *Let τ be a vocabulary that includes ordering. If* TIMES $\in \tau$ *then* BIT *is first-order definable. In particular,* FO($<$, BIT) = FO($<$, TIMES).

Proof We proceed with a series of definitions. First observe that the relations "x is the successor of y", "x divides y", "y is prime", and "y is a power of 2" can all be defined in first-order logic with $<$ and TIMES. These relations will be denoted respectively as $\mathrm{SUC}(x, y), \mathrm{DIVIDES}(x, y), \mathrm{prime}(y)$, and $p_2(y)$.

As an approach to defining $\mathrm{BIT}(x, i)$ we first define $\mathrm{BIT}'(x, y)$, which holds iff $2^i = y$ and $\mathrm{BIT}(x, i)$. For ease of notation, in what follows we treat the relations TIMES and SUC as functions. Note, however, that the relations can be reinserted with an extra quantification. In the proof given in [Imm88], BIT$'$ is defined as follows:

$$\mathrm{BIT}'(x, y) \equiv p_2(y) \wedge \exists u(2uy + y \leq x \wedge x < 2uy + 2y)$$

The new part of this proof is the observation that we can eliminate the use of $+$ in the above formula by factoring.

$$BIT'(x, y) \equiv p_2(y) \wedge \exists u(y(S(2u)) < x \wedge x < 2yS(u))$$

As the definition of BIT$'$ contained the only use of $+$ in the original proof, we can now continue following the exposition of Immerman.

BIT$'$ allows us to copy a sequence of bits. The following formula says that if $y = 2^i$ and $z = 2^j$ where $y > z$, then bit $i + j \ldots i$ of x are the same as bits $j \ldots 0$ of c.

$$\mathrm{COPY}(x, y, z, c) \equiv (\forall u. p_2(u) \wedge u \leq z)(\mathrm{BIT}'(x, yu) \leftrightarrow \mathrm{BIT}'(c, u))$$

All that remains to express BIT is to define the relation $2^i = y$. We can express this by encoding the following recurrence,

$$2^i = y \iff (\exists j)(\exists z. 2^j = z)(i = 2j + 1 \wedge y = 2z^2 \vee i = 2j \wedge y = z^2). \quad (7.1)$$

That is, by explicitly encoding a small power of 2, and then verifying the $\log i$ recursive claims of (7.1), we can express that $2^i = y$. We do this by asserting the existence of three variables, $Y, I,$ and Z. For each exponent k occuring in the recursion (7.1), we put a "1" in position k of Y, and write the binary representation of k in I, beginning from position k. Actually I only records the exponents k for which $\log k < k - k/2$, or $k > 2 \log k$, to ensure that no exponents overlap. We can explicitly encode the computations of (7.1) when the exponents cannot be recorded, as in this case $k \leq 3$. We mark the final bit of the binary representations written in I with a "1" placed in the same position of Z. Then we can extract the exponents

Table 7.1: Values of $Y, I,$ and Z to express $2^{15} = 32,768$

Position	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Y	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0
I	0	1	1	1	1	0	0	0	0	1	1	1	0	1	1	0	0
Z	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0

recorded in I with the relation PLACE that says that the exponent written in I beginning with position x is c:

$$\begin{aligned}\mathrm{PLACE}(I,x,c) \equiv \exists y\Big(y < x \quad &\wedge \quad \mathrm{BIT}(Z,y) \wedge \forall z\big(y < z < x \to \neg\mathrm{BIT}'(Z,z)\big)\\ &\wedge \quad \mathrm{COPY}(I,x,y,c)\Big)\end{aligned}$$

The role of the variables $Y, I,$ and Z is best illustrated with an example. To express the proposition $2^{15} = 32,768$ the variables $Y, I,$ and Z have the values given in table 1.

The leftmost "1" in Y is in the place of 2^{15}, and written beginning from position 15 in I is the exponent 15 written in binary, 1111. This arrangement allows us to encode the first computation in (7.1), namely that if y is the largest number satisfying $\mathrm{BIT}'(Y,y)$, in this case 32,768, and i is the leftmost number written in I, in this case 15, then $2^i = y$. Looking to the next "1" in Y in the place of 2^7 we can read the exponent 7 recorded in binary, 111, in I, and again we can use these positions to verify the computations of (7.1). Finally, there is a "1" at the place of 2^3 in Y, and exponent 3 written in binary from position 3 of I. This allows us to anchor the recursion by explicity verifying with TIMES that $2^3 = 8$.

Following the reasoning of this example, we construct a first-order formula which checks that $Y, I,$ and Z have the proper properties. Namely this formula says that if x is the largest number for which $\mathrm{BIT}'(y,x)$, then $x = y$.

$$\forall x\,\big(\mathrm{BIT}'(Y,x) \wedge \forall z\,\big(\mathrm{BIT}'(Y,z) \to z \leq x\big) \to x = y\big)$$

and that the leftmost number represented in I is i

$$\begin{aligned}\forall x_1 x_2\Big(\mathrm{BIT}'(I,x_1) \quad &\wedge \quad \forall z\big(\mathrm{BIT}'(I,z) \to z \leq x_1\big) \wedge \mathrm{BIT}'(Z,x_2)\\ &\wedge \quad \forall z\big(\mathrm{BIT}'(Z,z) \to z \leq x_2\big) \to \mathrm{COPY}(I,x_1,x_2,i)\Big).\end{aligned}$$

Then the formula checks that consecutive positions k, l in Y with a "1" are related by $2^k = 2^{2l+1}$ or $2^k = 2^{2l}$, and corresponding representations in I

satisfy $k = 2l + 1$ or $k = 2l$, accordingly:

$$\begin{aligned}\forall xy\Big(x < y \quad &\wedge \quad \mathrm{BIT}'(Y, y) \wedge \mathrm{BIT}'(Y, x) \wedge \forall z\big(y < z < x \rightarrow \neg\mathrm{BIT}'(Y, z)\big)\\ &\rightarrow \quad x = 2y^2 \wedge \forall cd\big(\mathrm{PLACE}(I, x, c) \wedge \mathrm{PLACE}(I, y, d) \rightarrow c = S(2d)\big)\\ &\vee \quad x = y^2 \wedge \forall cd\big(\mathrm{PLACE}(I, x, c) \wedge \mathrm{PLACE}(I, y, d) \rightarrow c = 2d\big)\Big).\end{aligned}$$

Finally, for the smallest exponent j recorded in I the formula explicitly hard-wires the value of 2^j. We know $j \leq 6$ because if $j > 6$ then j div $2 > 2\log(j$ div $2)$ and so would be recorded as well in I.

This gives the formula that defines the relation $\mathrm{EXP}_2(i, y)$ as desired, and at last we have our expression for BIT,

$$\mathrm{BIT}(x, i) \iff \exists y(\mathrm{EXP}_2(i, y) \wedge \mathrm{BIT}'(x, y)). \tag{7.2}$$

Corollary 7.1.2 PLUS *is first-order definable with* $<$ *and* TIMES

Proof By Thm (7.1.1), there is a first-order definition of BIT with TIMES and ordering. Formula (6.2) then gives a definition for PLUS in terms of BIT and ordering. ▮

A natural question to ask if ordering is necessary in the above corollary. The next theorem says that it is.

Theorem 7.1.3 TIMES *cannot first-order define* PLUS.

Proof Suppose to the contrary that $\phi(x, y, z) \in \mathrm{FO(TIMES)}$ defines PLUS. We define an automorphism f on $\langle n, \mathrm{TIMES}, 0, 1\rangle$ as follows. Write $x \in n$ as $x = 2^k 3^j y$, where y contains no factors of 2 or 3. Then we define $f : x \mapsto 2^j 3^k y$. This is an automorphism as $f(x_1 x_2) = f(2^{k_1+k_2} 3^{j_1+j_2} y_1 y_2) = 2^{j_1+j_2} 3^{k_1+k_2} y_1 y_2 = f(x_1) f(x_2)$. This map does not respect addition, however, as $f(2+2) = 9 \neg = 6 = f(2) + f(2)$. This contradicts the existence of ϕ as isomorphisms preserve the truth of first-order formulas. ▮

Corollary 7.1.4 *Ordering is not first-order definable with TIMES.*

Proof Suppose TIMES could define $<$. Then by corollary (7.1.2) TIMES could define PLUS, contradicting thm (7.1.3). ▮

7.2 FO($<$, PLUS) $\subset$ FO($<$, TIMES)

We have now shown FO($<$, PLUS) $\subseteq$ FO($<$, TIMES), as FO($<$, TIMES) $=$ FO(BIT) $=$ FO($<$, PLUS, TIMES) $\supseteq$ FO($<$, PLUS). The most direct way to show that in fact FO($<$, PLUS) $\subset$ FO($<$, TIMES) would be to find an example of a language L definable in FO($<$, TIMES) and then show using Ehrenfeucht-Fraïssé games that L is not definable in FO($<$, PLUS).

Unfortunately, we have not yet found a proof this direct. The proof instead goes by way of results relating to the Crane Beach Conjecture.

7.2.1 Crane Beach Conjecture

To introduce the Crane Beach Conjecture, we first must develop a few definitions. A *neutral letter* is a letter e that be inserted or deleted in a string without affecting the string's membership in a language L.

Definition Let A be an alphabet and $L \subseteq A^*$. A letter $e \in A$ is called neutral for L if for any $u, v \in A^*$, it holds that $uv \in L$ iff $uev \in L$.

As an example, 0 is a neutral letter for PARITY. As membership in PARITY only depends on the number of ones a string contains, zeros can be arbitrarily inserted to no effect. On the other hand, the language $\{0^n 1^n : n \in N\}$ does not have a neutral letter in $\{0, 1\}$ as membership in this language depends on a very specific numerical relationship between the positions of zeros and ones in the string.

The idea behind considering languages with neutral letters is that, in a language with a neutral letter, only the relative positions of the letters should matter and not any more complicated numerical relationship, as in $0^n 1^n$. This line of thinking led to the following conjecture.

Definition (Crane Beach Conjecture) Let $\mathcal{N}$ be a set of numerical predicates. We say the Crane Beach Conjecture is true for $\mathcal{N}$ iff every language $L \in$ FO($<, \mathcal{N}$) that has a neutral letter is also definable in FO($<$).

A recent paper on the Crane Beach Conjecture [BIL+01] shows it is true for ($<$, PLUS), but false for ($<$, PLUS, TIMES). This means that there exists a language, definable in FO($<$, PLUS, TIMES), which violates the Crane Beach Conjecture, and thus is not definable in FO($<$, PLUS). Together with FO($<$, PLUS, TIMES) = FO($<$, TIMES) from Thm (7.1.1) this gives

Corollary 7.2.1 FO($<$, PLUS) $\subset$ FO($<$, TIMES)

The proof that the Crane Beach Conjecture is true for ($<$, PLUS) still uses Ehrenfeucht-Fraïssé games, but in a more indirect way. The proof proceeds by contradiction, assuming that there is a language L definable in FO($<$, PLUS) but not in FO($<$). If L is not definable in FO($<$), then there exists some k such that for any strings $u \in L$ and $v \notin L$, Duplicator has a winning strategy in the k-round game for $(<)$ on u, v. This winning strategy for Duplicator with respect to $<$ is then expanded into a winning strategy for Duplicator in the game with respect to $(<, \mathrm{PLUS})$. This implies that $L \notin \mathrm{FO}(<, PLUS)$, a contradiction.

7.3 FO(<, TIMES) = FO(<, DIVIDES)

Define the predicate DIVIDES(x, y) to mean that x divides y. It is easy to see that TIMES can define DIVIDES with the formula

$$\mathrm{DIVIDES}(x, y) \iff \exists z \mathrm{TIMES}(z, x, y)$$

Here we show that DIVIDES itself actually has quite a bit of expressive power, and together with ordering can define TIMES. First, we show DIVIDES can express simpler predicates like "z is the least common multiple of x and y", denoted LCM(x, y, z), and "z is the greatest common divisor of x and y", denoted GCD(x, y, z)

$$\begin{aligned} \mathrm{LCM}(x, y, z) &\equiv (x|z \wedge y|z \wedge \forall z'(x|z' \wedge y|z' \rightarrow z \leq z')) \\ \mathrm{GCD}(x, y, z) &\equiv (z|x \wedge z|y \wedge \forall z'(z'|x \wedge z'|y \rightarrow z' \leq z)) \end{aligned}$$

The key of our approach to defining TIMES is the fact that if the GCD of x and y is 1, that is if x and y are relatively prime, then the LCM of x and y is their product, LCM(x, y, xy).

Theorem 7.3.1 *Let τ be a vocabulary that includes ordering. If* DIVIDES $\in \tau$ *then* TIMES *is first-order definable. In particular,* FO$(<, \mathrm{DIVIDES}) =$ FO$(<, \mathrm{TIMES})$.

Proof Let $x, y \in n$. We wish to express the product xy using DIVIDES and ordering. First we break x into two parts z_1 and z_2 such that $x = z_1 z_2$ and z_1 is the largest factor of x relatively prime to y.

Claim 7.3.2 *z_1 and z_2 are relatively prime.*

Proof Suppose p is a prime dividing both z_1 and z_2. Either p divides y or p does not divide y. If p divides y then p cannot divide z_1 as z_1 and y are relatively prime. On the other hand, if p does not divide y then $z_1 p$ would be relatively prime to y, contradicting the defininition of z_1. ▮

We also need to check that z_1, z_2 can be so defined.

$$\exists z_1(z_1|x \wedge \mathrm{GCD}(z_1, y, 1) \wedge \forall z_1'(z_1'|x \wedge \mathrm{GCD}(z_1', y, 1) \rightarrow z_1' \leq z_1))$$

$$\exists z_2(\mathrm{GCD}(z_1, z_2, 1) \wedge \mathrm{LCM}(z_1, z_2, x))$$

We have now reduced the problem to defining the product z_2y. As z_1 is relatively prime to z_2 and relatively prime to y, it is also relatively prime to the product z_2y. Thus if we can define z_2y, then we can define z_1z_2y as $\mathrm{LCM}(z_1, z_2y)$.

Now z_2 does not necessarily divide y, but because of our definition of z_1, every prime divisor of z_2 is also a prime divisor of y.

Claim 7.3.3 *z_2 and $y+1$ are relatively prime*

Proof Assume there exists a prime $p > 1$ such that $p|z_2$ and $p|y+1$. But as p is a factor of z_2, it is also a factor of y, thus $p|y$. Then $p|(y+1-y)$ which implies that p divides 1, a contradiction. ∎

The same reasoning can be used to show $\mathrm{GCD}(z_2, y-1, 1)$. Thus as z_2 is relatively prime to $y-1$ and $y+1$, we can define the products $z_2(y-1) = \mathrm{LCM}(z_2, y-1)$ and $z_2(y+1) = \mathrm{LCM}(z_2, y+1)$. We use these products to bookend the product z_2y.

Claim 7.3.4 $z_2y = t \iff z_2(y-1) < t < z_2(y+1) \wedge z_2|t$

Proof $\Rightarrow$ $z_2(y-1) < z_2y < z_2(y+1)$ and $z_2|z_2y$

$\Leftarrow$ Since $z_2|t$, there is some k such that $t = z_2k$. By the other condition, we have

$$z_2(y-1) < z_2k < z_2(y+1).$$

Assuming z_2 is not 0, which could only happen if $x = 0$, in which case we would not be here, then

$$y-1 < k < y+1.$$

Thus $k = y$, and so $t = z_2y$. ∎

Chapter 8

Conclusions and Further Directions

Although very few strict containments are known amongst larger complexity classes, we now know the following strict containments and equivalences in small extensions of first-order logic:

$$\begin{aligned} \mathrm{FO} \subset \mathrm{FO}(<) \subset \mathrm{FO}(<,\mathrm{PLUS}) \subset \mathrm{FO}(<,\mathrm{TIMES}) &= \mathrm{FO}(<,\mathrm{DIVIDES}) \\ &= \mathrm{FO}(<,\mathrm{BIT}) \end{aligned}$$

There are many other combinations we could consider, and also some predicate combinations whose complexities are orthogonal to each other — each not being able to first-order define the other.

One particularly interesting orthogonal relationship is with ordering. Ordering has a very special role in descriptive complexity, and is at the root of some of its deepest problems such as finding a language for order independent **P**. One of the most elegant gifts of descriptive complexity is that we can represent problems by predicates and so not worry about the particulars of how a problem is encoded. But this also gives rise to the 'order mismatch' when comparing with structural complexity classes. When inputs are encoded, for example on a Turing Machine, they inevitably acquire and ordering, simply by their position on the input tape.

Although ordering seems like a very simple relation, its complexity is actually orthogonal to many more powerful predicates. We have seen that it cannot be defined by TIMES. We conjecture as well that ordering cannot be defined by BIT.

In chapter 4, we sketched a general game method for proving inexpressibility results in the hope that this method could be used to separate larger complexity classes. It would be nice to know if there is a limit to how far this proof method can go in separating complexity classes, for example, a proof that the game method cannot separate **P** from **NP**. It does not seem like the two techniques we are aware of to prove limits on techniques, showing that they relativize or are natural, can be directly applied to the game technique.

Although it is over 50 years old now, the paper of Julia Robinson [Rob49] is still very much alive. In this thesis we showed that her two main definability results, that multiplication and successor can define addition, and that the divisibility relation and successor can define multiplication, also both hold in the finite case (where ordering replaces successor). In the same paper, Julia Robinson suggested a further definability problem which was left open. The problem, now termed Robinson's problem, is whether the relatively prime relation and successor can first-order define multiplication. Later it was shown in the PhD thesis of Alan Woods [Woo81] that a positive solution to Robinson's problem is equivalent to the existence of a $k \in N$ such that for all pairs (x, y), if $x + i$ and $y + i$ have the same prime divisors for all $0 \leq i \leq k$ then $x = y$. This last statement is a weakening of a conjecture made by Erdös, now called the Woods-Erdös conjecture.

Determining the expressive power of the relatively prime predicate would also be informative in elucidating the relationship between three concepts:

- Crane Beach Conjecture: the new conjecture is that the Crane Beach Conjecture is true of a set of predicates $\mathcal{N}$ iff $\mathcal{N}$ cannot count beyond a constant.
- Jump in Counting: Is there a set of 'natural' predicates $\mathcal{N}$ which can count beyond a constant but which cannot define PLUS and TIMES?
- Gap Between Decidability and Undecidability: For what set of predicates $\mathcal{N}$ is the theory $\langle \omega, \mathcal{N} \rangle$ undecidable yet unable to express all recursively enumerable sets of finite relations over ω (that is, unable to define PLUS and TIMES)

Also in his thesis [Woo81], Woods shows that the theory $\langle \omega, \ ', \perp \rangle$ is undecidable, and that multiplication can be defined with bounded $(<, \perp)$ formulas over ω. In fitting with the building body of evidence associating these ideas, we conjecture that $(<, \perp)$ can define TIMES on finite structures, and that $(<, \perp)$ is false for the Crane Beach Conjecture.

Bibliography

[AF90] M. Ajtai and R. Fagin. Reachability is harder for directed than for undirected finite graphs. *The Journal of Symbolic Logic*, 55:113–150, 1990.

[BIL$^+$01] D. Barrington, N. Immerman, C. Lauteman, N. Schweikardt, and D. Thérien. The crane beach conjecture. pages 187–196. LICS '01, 2001.

[BIS90] David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990. Available at `citeseer.nj.nec.com/barrington90uniformity.html`.

[Bus91] S. Buss. The graph of multiplication is equivalent to counting. Available from `ftp://euclid.ucsd.edu/pub/sbuss/research`, 1991.

[CDL] A. Chiu, G. Davida, and B. Litow. NC^1 division. Manuscript, 1999.
Available from the website `http://www.cs.jcu.edu.au/ bruce` as `/papers/crr00_3.ps.gz`.

[Fag74] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In R. Karp, editor, *Complexity of Computation, SIAM-AMS Proceedings*, volume 7, pages 43–73, 1974.

[Fag75] R. Fagin. Monadic generalized spectra. *Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik*, 21:89–96, 1975.

[FSS84] M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17:13–27, 1984.

[Hes01] W. Hesse. Division is in uniform TC0. ICALP 2001: Twenty-Eighth International Colloquium on Automata, Languages, and Programming, 2001.

[Imm82] N. Immerman. Relational queries computable in polynomial time. *14th Symposium on Theory of Computation*, pages 147–152, 1982.

[Imm87] Neil Immerman. Languages that capture complexity classes. *SIAM Journal of Computing*, 16(4):760–778, 1987. available at citeseer.nj.nec.com/89379.html.

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal of Computing*, 5:935–938, 1988.

[Pre29] M. Presburger. Über die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchem die addition als einzige operation hervortritt. In *Comptes-rendus du I Congrès des mathématiciens des Pays Slaves*, pages 92–101 and 395, 1929.

[Rob49] J. Robinson. Definability and decision problems in arithmetic. *The Journal of Symbolic Logic*, 14:98–114, 1949.

[TMR53] A. Tarski, A. Mostowski, and R. Robinson. *Undecidable Theories.* North-Holland, Amsterdam, 1953.

[Var82] M. Vardi. Complexity of relational query languages. *14th Symposium on Theory of Computation*, pages 137–146, 1982.

[Woo81] A. Woods. *Some problems in logic and number theory and their connections.* PhD thesis, University of Manchester, Manchester, 1981.