REGULAR QUANTIFIERS IN LOGICS

by

Sreejith A V

The Institute of Mathematical Sciences
Chennai 600113

A thesis submitted to the
Board of Studies in Mathematical Sciences

In partial fulfilment of requirements
For the Degree of

DOCTOR OF PHILOSOPHY
of
HOMI BHABHA NATIONAL INSTITUTE

December 16, 2013

Homi Bhabha National Institute

Recommendations of the Viva Voce Board

As members of the Viva Voce Board, we recommend that the dissertation prepared by **Sreejith A V** titled "Regular quantifiers in logics" may be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

_____ **Date :**

Chairman and Convener: Kamal Lodaya

_____ **Date :**

Member: Antoine Meyer

_____ **Date :**

Member: Meena Mahajan

_____ **Date :**

Member: R Ramanujam

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to HBNI.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it may be accepted as fulfilling the dissertation requirement.

_____ **Date :**

Guide: Kamal Lodaya

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fullfillment of requirements for an advanced degree at Homi Bhabha National Institute (HBNI) and is deposited in the Library to be made available to borrowers under rules of HBNI.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgement of source is made. Request for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the Competent Authority of HBNI when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

Sreejith A V

DECLARATION

I, hereby declare that the investigation presented in the thesis has been carried out by me. The work is original and has not been submitted earlier as a whole or in part for a degree/diploma at this or any other Institute/University.

Sreejith A V

Abstract

First order logic on words is a well-studied logic [Tho97, EFT94]. It is known that first order logic (FO) with a linear order, FO[<] (in our notation, we enclose the relations used in square bracket) cannot count more than a constant [Lib04]. From the result of Furst, Saxe and Sipser [FSS84] we know that, the logic FO[<, Arb] cannot define the language

$$L_{even} = \{w \in \{0,1\}^* \mid \text{ number of 1 s is even in } w\}$$

Here Arb stands for arbitrary numerical predicates (predicates whose truth depends only on the position of the variables in the linear order and not on the letters at these positions). This shows that first order logic cannot count modulo a number (two in the language L_{even}). This inability to count, led to extending first order logic by different kinds of quantifiers [STT95, Imm86, Ruh99b].

In this thesis, we study logic on words extended with *regular* quantifiers. Modulo counting quantifiers are one particular example of such quantifiers, which have been well studied in the past [STT95, Pot94, RS07]. These quantifiers can be generalized to group quantifiers and further to monoid quantifiers [BIS90, Str94], all being regular quantifiers.

The logics we extend can be classified into two parts. In the first part, we look at logics which define regular languages like FO[<] and linear temporal logic (LTL). We extend these logics with the above mentioned regular quantifiers. In the second part, we look at regular quantifiers over a linear order and an addition function which respects the linear order. This takes us outside regular languages. We ask the following questions about the logics we consider.

- Expressiveness: We investigate the languages definable in these logics.
- Satisfiability and model checking: The complexity of satisfiability and model checking for these logics are studied.

The title of the thesis comes from the observation that the quantifiers we consider, in the presence of a linear order, can only define regular languages [Str94]. There are quantifiers like the unary counting quantifiers [GOR97, Sch05] which can define non-regular languages.

In the first part of our work, we show that LTL extended with modulo counting/ group operators (LTL$_{\text{GRP}}$) and FO[$<$] extended with modulo counting/group quantifiers (FO$_{\text{GRP}}$[$<$]), both accept the same set of languages. We then go on to show that the satisfiability and model checking for LTL$_{\text{GRP}}$ is PSPACE-complete. We also look at satisfiability of various fragments of this logic. Then we show that the two variable fragment of FO$_{\text{GRP}}$[$<$] is EXPSPACE-complete. We also analyse certain important sublogics.

In the second part of our work, we study first order logic with a linear order and the arithmetic predicate, $+$. We first show that the two variable fragment of FO$_{\text{MOD}}$[$<,+$] is undecidable. Then we show that over a unary alphabet satisfiability of FO$_{\text{MOD}}$[$<,+$] is 2EXPSPACE. Finally we investigate the expressive power of $\mathcal{M}[<,+]$, where $\mathcal{M}$ is a set of monoid quantifiers. We show, using the concept of a neutral letter [BIL$^+$05], that the class of neutral letter languages definable in $\mathcal{M}[<,+]$ is equivalent to those definable in $\mathcal{M}[<]$. Using the above claim, we are able to show that the logics $\mathcal{M}_1[<,+]$ is different from $\mathcal{M}_2[<,+]$, if the set of monoid quantifiers $\mathcal{M}_1$ and $\mathcal{M}_2$ are different. This lets us answer a conjecture of Roy and Straubing [RS07] that FO[$<,+$] and MOD[$<,+$] are incomparable. We also show that given a regular language L, it is decidable whether L is definable in MOD[$<,+$] or not.

Acknowledgements

I would like to express my sincere gratitude to my advisor Professor Kamal Lodaya for the continuous support of my research, for his motivation, patience, enthusiasm and immense knowledge. His guidance helped me to proceed through the doctoral program and complete my thesis. He gave me the freedom to say no and I specially thank him for that.

Besides my advisor, I would like to thank the other faculty members Prof. Ramanujam, Prof. Meena and Prof. Arvind, for their encouragement, support and helpful suggestions.

I thank the Institute of Mathematical Sciences (IMSc), for providing me with financial assistance during my research career. It has given me a truly wonderful atmosphere and facilities for pursuing my research. I thank all the faculty members, the office staff, library staff and the students for their cooperation during my stay at IMSc.

I would also like to thank University of Tübingen for giving me an opportunity to visit them and also for the financial support. Special thanks to Andreas, Klaus-Jörn and Renate.

I would like to thank the anonymous referees for pointing out improvements/mistakes in the thesis.

I am especially grateful to Ramchandra and Baskar. They were of immense support not only academically but also for all the extra curricular fun we had. Baskar requires special mention for giving me company whenever I went out for food. A special thanks to Mubeena, my officemate for all the help and advice. Whenever I wanted to watch a movie, she would get it for me.

I had lots of interesting political/philosophical discussions with Sudhir and Neeraj. The never ending "stupid" talks we had were always entertaining. Learning the basics of Tennis with them was also fun. I would also like to thank Yadu, Karteek, Baskar, Ramchandra with whom I had lots of TT and badminton sessions.

Special thanks to Philip, Manju, Anoop, Narayanan, Madhushree, Prajakta, Rohan, Srikanth, Nutan, Raja, Kunal for making my stay in IMSc fun filled. Outside IMSc, Yousuf, Ajesh, Karthik, Mathew and Deepu were a joy to hang out with.

Last but not the least I would like to thank my parents Ajithkumar and Vijayalekshmi, sister Sreeja and my grandparents for supporting me in this long journey. The thesis is as much a product of their effort as mine.

Publications

Kamal Lodaya and A.V. Sreejith. LTL can be more succinct. In *Proc. 8th ATVA, Singapore*, volume 6252 of *LNCS*, pages 245–258, 2010.

A. V. Sreejith. Expressive completeness for ltl with modulo counting and group quantifiers. *Electronic Notes in Theoretical Computer Science*, 278:201–214, 2011.

Andreas Krebs and A. V. Sreejith. Non-definability of languages by generalized first-order formulas over ($\mathbb{N}$, $+$). In *LICS*, pages 451–460, 2012.

CONTENTS

Abstract i

Acknowledgements i

Publications i

Contents i

List of Figures v

List of Tables vi

I INTRODUCTION 1

1 OVERVIEW 2

1.1 Results 4

1.1.1 Formal Languages 4

1.1.2 Presburger arithmetic 5

1.1.3 Verification 5

1.1.4 Circuit Complexity 6

1.2 Organization of the thesis 7

2 PRELIMINARIES 8

2.1 Basic Definitions 8

2.2 Complexity classes 9

2.3 Semigroups 10

2.4 Linear temporal logic 12

2.4.1 Modulo counting operators 12

2.4.2 Group operators 14

2.4.3 Extended Temporal Logics 16

2.4.4 Other definitions 18

2.5 First order logic 19

2.5.1 Counting quantifiers . . . 20
2.5.2 Monoid/Group quantifiers . . . 21
2.5.3 Extended first order logic . . . 22
2.5.4 Few examples . . . 23

II REGULAR LANGUAGES 24

3 SURVEY ON REGULAR LANGUAGES 25
3.1 Expressiveness . . . 25
3.1.1 Group quantifier Extensions . . . 26
3.1.2 A sublogic . . . 27
3.2 Satisfiability and model checking . . . 29

4 LTLGRP EXPRESSIVENESS (EQUIVALENCE WITH FOGRP[<]) 31
4.1 Introduction . . . 31
4.2 Properties of LTLGRP . . . 31
4.2.1 Separation property of LTLGRP . . . 32
4.2.2 Expressive Completeness of LTLGRP . . . 37
4.3 UTL and two variable fragment of FO[<] . . . 39
4.4 Discussion . . . 41

5 LTLGRP SATISFIABILITY 43
5.1 Introduction . . . 43
5.2 Modulo counting . . . 44
5.2.1 A PSPACE upper bound . . . 44
5.2.2 Corresponding lower bound . . . 45
5.3 Length modulo counting . . . 47
5.3.1 Length modulo counting - Upper bound . . . 48
5.3.2 Length modulo counting - Lower bound . . . 51
5.4 Satisfiability of TL[DUR] logic . . . 53
5.5 Discussion . . . 55

6 FO[<] SATISFIABILITY 57

6.1 Introduction . . . 57
6.2 Upper bounds via linear temporal logic . . . 57
6.3 Lower bounds via tiling problems . . . 61
6.3.1 Tiling problems . . . 61
6.3.2 Modulo counting is EXPSPACE-hard . . . 62
6.3.3 Modulo predicates is NEXPTIME-hard . . . 65
6.4 Discussion . . . 65

III LOGICS WITH ADDITION 67

7 SURVEY ON ADDITION RELATION 68
7.1 Expressiveness . . . 68
7.1.1 Descriptive Complexity of Circuit classes . . . 69
7.1.2 The Crane Beach conjecture . . . 71
7.1.3 Presburger arithmetic extended with modulo counting . . . 73
7.2 Satisfiability . . . 74

8 FOGRP$[<,+]$ EXPRESSIVENESS (LOWER BOUNDS RESULTS) 76
8.1 Introduction . . . 76
8.2 Results . . . 76
8.2.1 Non definability Results . . . 76
8.2.2 Decidability of Regular languages in $L_{\mathcal{S}}[<,+]$ . . . 78
8.3 Proof Strategy . . . 80
8.4 Proof of the Main Theorem . . . 82
8.4.1 Definitions . . . 82
8.4.2 The Proof . . . 83
8.5 Proof of Lemma 8.4.6 . . . 87
8.5.1 Intervals and Linear Functions . . . 87
8.5.2 Treating each B_t differently . . . 93
8.5.3 Sorting Tree . . . 96
8.5.4 Constructing the active domain formula . . . 102

8.6 Discussion 105

9 FOGRP[<, +] SATISFIABILITY 107

9.1 Introduction 107

9.2 Two variable logic with addition 107

9.3 Presburger arithmetic with mod quantifiers 110

9.4 Discussion 115

IV CONCLUSION 116

10 FUTURE DIRECTIONS AND OPEN QUESTIONS 117

Bibliography 118

Index 127

LIST OF FIGURES

Figure 1 Automaton for S_5 . . . 17
Figure 2 Star free Languages - Different characterizations . . . 25
Figure 3 Logics with group quantifiers . . . 27
Figure 4 Unambgious Languages - Different characterizations . . . 28
Figure 5 Past Group modality . . . 32
Figure 6 Timeline for $a\check{S}b \wedge \beta$ . . . 33
Figure 7 Timeline for $G^P_{g'}\langle ...\rangle$ . . . 34
Figure 8 Regular quantifiers and Logics . . . 41
Figure 9 Proof of removing addition . . . 81
Figure 10 Boundary points and intervals . . . 89
Figure 11 Positions of interest in a word . . . 91
Figure 12 Boundary points and group values . . . 93
Figure 13 Active Domain . . . 97
Figure 14 Sorting tree . . . 99
Figure 15 Regular quantifiers and addition: Expressiveness . . . 105

LIST OF TABLES

Table 1 Satisfiability of FO[<] 30

Table 2 LTL$_{\text{GRP}}$ to FO$_{\text{GRP}}$[<] 39

Table 3 Expressive power of various fragments of LTL$_{\text{GRP}}$ 42

Table 4 LTL and extensions: Satisfiability 43

Table 5 Two variable logic: Satisfiability 66

Table 6 Generalized quantifiers and Expressiveness 70

Part I

INTRODUCTION

1

OVERVIEW

In the thesis, the author views **Logic** as a *formal system* which consists of a finite set of "base properties" (called *propositional symbols* or just *propositions*) along with certain simple rules to construct more and more complicated properties. In particular, we will be looking at *properties on words*. For example, the "set of all words of even length formed from the letters $\{a, b\}$", is a property on words. The logic we consider will be first order logic on *word* structures. The structure will have built-in predicates like linear order $<$, successor (*suc*) relation, and also the arithmetic predicate $+$. These logics have been well studied in the past. For example, the logic FO[$<$] (first order logic with a linear order relation, we use the conventional notation of putting the relations in square brackets) has other equivalent characterizations. The set of star free languages [SM73], the set of languages recognized by aperiodic monoids [Sch65], the set of languages definable by linear temporal logic [Kam68] all exactly characterize the set of languages definable by FO[$<$].

Limitations of first order logic

The major limitation of first order logic is its inability to *count* more than a constant. It is a well known fact that the logic FO[$<$] cannot count more than a constant [Lib04]. One might wonder, whether adding new *relations* could help in counting. Extra relations can help, but not much. The logic FO[$<, +, *$] can count upto a polynomial in the logarithm of the input size [ABO84, FKPS85, RW91]. On the other hand, from the result of Furst, Saxe and Sipser [FSS84] we know that even the logic FO[$<$, Arb] (here *Arb* stands for arbitrary predicates) cannot count upto any polynomial in input size. For example, the following language is not expressible in FO[$<$, Arb]

$$\{w \in \{a, b\}^* \mid \text{number of } a \text{ in } w > \text{ number of } b \text{ in } w\}$$

The above language requires the logic to count the number of occurrences of a (this count can go upto $\frac{n}{2}$, where n is the size of the word) and then verify whether it is greater than half the size of the input. The counting required can be shown to be greater than any polynomial in the logarithm of n. Let us look at yet another example. Here we require a different kind of counting. Furst, Saxe and Sipser [FSS84] showed that the following language is also not definable in FO[$<$, Arb].

$$\mathcal{L}_a = \{w \in \{a, b\}^* \mid \text{ there are an even number of occurrences of letter } a \text{ in } w\}$$

The above language is regular (a DFA of size 2 can recognize this language). The inexpressibility of $\mathcal{L}_a$ shows the inability of FO[<, Arb] to count (modulo 2).

Extensions

We observed the severe limitations of first order logic in expressing some simple properties. We also saw that, adding extra relations cannot help much. This has led to attempts at incorporating additional *quantifiers* into these logics, which hopefully can increase the expressive power. Below we give a few attempts which have been made so far.

- Straubing, Thérien and Thomas [STT95], looked at first order logic over arbitrary relations extended with modulo counting quantifiers. The quantifiers can count the number of positions (modulo some number) a particular formula is satisfied in a word.
- Barrington, Immerman and Straubing [BIS90] looked at logics with group quantifiers, which generalized modulo counting to operations in a finite group. They show that these logics are closely related to well known circuit complexity classes.
- Baziramwabo et al [BMT99], extended linear temporal logic with group quantifiers and looked at algebraically characterizing these classes.
- Ruhl [Ruh99b], showed that over an ordered finite structure, first order logic with unary counting quantifiers (here the quantifiers can count the number of positions a particular formula is satisfied and compare it with a variable) cannot express many properties. For example, Ruhl proved that multiplication is not definable in FO[< , +] even with counting quantifiers.
- Immerman [Imm86] and Vardi [Var82] extended first order logic with a least fixed point, using it to characterize the class P over ordered structures.

Logics considered

In this thesis, we thoroughly investigate *Modulo counting* and *Group* quantifier extensions of various logics. The logics we consider can be broadly classified into two parts.

PART 1: INSIDE REGULAR LANGUAGES

> *Here we look at first order logic with a linear ordering and linear temporal logic. These logics can only express regular languages. We also look at various fragments of these logics. In particular the two variable logics are looked at.*

PART 2: LOGICS WITH ADDITION PREDICATE

> *Here we look at first order logic with a linear order and an addition relation faithful to the linear order. These logics can express non-regular languages. For example, the language $\{a^n b^n \mid n \in \mathbb{N}\}$ is definable in this logic.*

We then study the consequences of extending each of the above logics with modulo counting and group quantifiers.

Questions asked

We ask the following two questions about the logics we consider.

THE EXPRESSIVE POWER OF THE LOGICS

> *We investigate the expressive power of the logics extended with the modulo counting and group quantifiers.*

SATISFIABILITY OF THE LOGICS

> *Here we investigate the complexity of satisfiability and model checking of the extended logics. We try to pinpoint the exact complexity of these problems.*

We will broadly refer to them as "expressiveness" and "algorithmic" questions respectively.

1.1 RESULTS

Our results can be broadly classified under the following subdivisions.

1.1.1 *Formal Languages*

The satisfiability of first order logic (with $<$ relation) and its various sublogics, especially the two variable fragment of it, has been well studied in the past [SM73, Sto74, EVW02]. We look at first order logic extended with modulo counting quantifiers. Straubing and Therien in [STT95] studied the expressiveness of this logic. The question of satisfiability has been left unanswered.

In particular, we show that the satisfiability of $\mathrm{FO}^2\mathrm{MOD}[<]$ is in EXPSPACE. We also show a sublogic to be EXPSPACE-hard. On the other hand, we show that unary counting makes it undecidable. We also identify a sublogic which is NEXPTIME-complete. Finally we look at the logic with an addition relation. It is easy to show that $\mathrm{FO}[<, +]$ is undecidable. We strengthen this to show that the satisfiability of $\mathrm{FO}^2[<, +]$ is also undecidable.

1.1.2 *Presburger arithmetic*

By the results of Ginsburg and Spanier [GS66], Ruhl [Ruh99b], Schweikardt [Sch05] we know that *Presburger arithmetic* is closed under unary counting quantifiers. Hence we can observe that the logic is also closed under modulo counting quantifiers. Here we look at the satisfiability of Presburger arithmetic extended with modulo counting quantifiers. We show that a 2EXPSPACE machine can check for truth of a given sentence.

1.1.3 *Verification*

Let us look at a slightly different logic, Linear temporal logic (LTL). It is known [Kam68] that it is expressively equivalent to FO[<]. The advantage of LTL over FO[<] is that there is an efficient reduction from LTL to automata. This permits faster algorithmic verification of this logic. Hence LTL has found wide use as a specification language in industry.

Due to the fact that LTL and FO[<] are expressively equivalent, all the limitations we saw for FO[<] also carry over to LTL. Hence linear temporal logic cannot define properties which involve counting. Therefore, the language $\mathcal{L}_a$ is not definable in LTL. On the other hand, these are some of the properties which comes often in verification but are not definable in LTL. Let us look at an example where periodic notion of time is useful.

Example 1.1.1. Calendars:
They are defined [SN92] as a periodic set of consecutive intervals which partition time. Such a periodic interval is called the granularity *of the calendar. For example, "every week", "every month" etc are calendars.*

This notion helps us to state properties like "every monday", the formula ϕ holds, or "every hour" the school bell should ring. These are properties which requires a periodic notion of time. Many extensions to LTL have been proposed in literature to express such properties [Dem06], [SN92]. Let us look at a concrete example.
We want to state that "every monday"at $7:00$ am, the formula ϕ holds. Let us assume that, each state represents an hour. Then we know that after every 24 states, it is next day. Let us assume that "time" starts at Monday $0:00$ am. Then we can state "every 24 hours", the formula ϕ should hold, in our logic as follows:

$$\mathrm{G}\left(\mathrm{MOD}^{P}_{7,24} true \Rightarrow \phi\right)$$

The formula states that, whenever we see a state which is at a distance of $7(\mod 24)$ from the initial state, then that state should also satisfy the formula ϕ (Refer to Preliminaries for the exact meaning of the notations).

In the above example we counted the number of states (modulo 24). Our logic, infact can count more. It can count the number of states having a certain property. For example

we can state that after "every 5 occurrences of states which satisfy ϕ", we should send a "signal". We can write this as follows.

$$\mathrm{G}\left(\mathrm{MOD}^{P}_{0,5}\phi \Rightarrow \ SentSignal\right)$$

Our first result is a new proof of the expressive equivalence of LTL when extended with modulo counting (and group operators) and FO[<] extended with the corresponding modulo counting quantifiers (and group quantifiers). It is also known that any regular language can be expressed in first order logic with group quantifiers and vice versa. Thus we have a temporal logic, LTL with group operators, LTLGRP which can express any regular language. We then look at the algorithmic questions. We show that both the satisfiability and model checking problems for LTLGRP are in PSPACE. Thus we have a temporal logic which can express any regular language and which has the nice properties required for verification purposes. We also identify certain fragments of this logic which lie in complexity classes below PSPACE. For example, we identify that the satisfiability of the logic, TL[F,LEN] (a sublogic of LTLGRP) is in Σ^{P}_{3}(a complexity class between NP and PSPACE and believed to be different from both). It is known that a fragment of LTL known as UTL is equivalent to $\mathrm{FO}^2[<]$. We extend this and prove that UTL (the sublogic of LTL without the until operator, See Chapter 2 Section 2.4) with modulo counting (and group operators) is equivalent to $\mathrm{FO}^2[<]$ with corresponding modulo counting (and group) quantifiers. We also show that the satisfiability of UTL with modulo counting operator is PSPACE-hard.

1.1.4 *Circuit Complexity*

It is known [Imm87b] that the set of languages accepted by dlogtime-uniform AC^0 circuits is exactly those definable in $\mathrm{FO}[<,+,*]$ (here $*$ stands for multiplication). The logic which uses only modulo counting quantifiers, $\mathrm{MOD}[<,+,*]$ on the other hand captures exactly the circuit complexity class CC^0. Similarly other finite monoid quantifiers capture other circuit classes. One of the biggest open problems in circuit complexity is the separation of these circuit classes. For example, it is not known whether there exists a language which is in AC^0, but not in CC^0. The above model-theoretic view of these circuit classes helps us to look at these separation questions from the perspective of logic. The hope is to bring lower bound techniques available in logic, like EF-games to prove non-definability of some languages in the circuit classes. We give a more detailed introduction in Chapter 8.

As a first step in this direction, we ask the question of separating these logics when the multiplication relation is not present. That is, can we separate $M_1[<,+]$ from $M_2[<,+]$, where M_1 and M_2 are different sets of group quantifiers. We give a general technique to prove lower bounds for logics of the form $M_1[<,+]$, which can then be used to separate $M_1[<,+]$ from $M_2[<,+]$. For example, we are able to show that $MOD[<,+]$ is incomparable to $FO[<,+]$. To show lower bounds we look at "neutral letter languages", which are languages with a neutral letter, i.e a letter which can be removed or inserted into any word without affecting its membership in the language.

1.2 ORGANIZATION OF THE THESIS

We organize the thesis as follows. Every chapter ends with a Discussion section, where we give a summary of the results of the chapter and also look at open questions and directions for future research. We divide the thesis into parts which address questions inside regular languages and those outside.

The thesis is organized into four parts. Part i gives an introduction to the thesis. Chapter 1 looks at the questions we address and the motivation for these questions. This is followed by Chapter 2 which gives the Preliminaries required for the rest of the book. We give a basic introduction to certain logics and also introduces common notations used throughout the thesis.

Part ii looks at logics which can define only regular languages. As we mentioned earlier, here we look at first order logic with a linear ordering and linear temporal logic. We begin this part by Chapter 3. The chapter gives a research survey of the area, relevant to our work. Chapter 4 shows that LTL extended with group operators is equivalent to FO[<], extended with group quantifiers. The chapter, also looks at the two variable fragment of first order logic and the UTL logic. We also identify connections between alternating finite automatas and modulo counting. Chapter 5 looks at the satisfiability of linear temporal logic with group operators. It also considers various fragments of this logic. Chapter 6 looks at the satisfiability of the two variable fragments of first order logic and various counting quantifiers. We show that $\mathrm{FO}^2[<]$ with modulo counting is EXPSPACE-complete, whereas $\mathrm{FO}^2[<]$ with arbitrary counting is undecidable.

Part iii looks at first order logic with a linear order and addition relation. Again, we begin by a Chapter 7 giving a survey of the relevant area. This is followed by Chapter 8 giving a method to prove lower bound results for the logic $M[<,+]$, where M is a set of monoid quantifiers. This is then used to show that logics using different group quantifiers characterize different class of languages. Chapter 9 looks at Presburger arithmetic extended with modulo counting quantifiers. We show that satisfiability of this logic is 2EXPSPACE. In this chapter, we also show that satisfiability of $\mathrm{FO}^2[<,+]$ is also undecidable.

In the final part iv we conclude the thesis. Chapter 10 gives the summary of the results of the thesis. We also look at the questions left open in the thesis and also give suggestions and directions for future work.

2

PRELIMINARIES

In this chapter we present the necessary definitions and notations used in this thesis. We will introduce linear temporal logic and first order logic. Moreover, we point out the important connections between these logics and semigroups.

2.1 BASIC DEFINITIONS

We denote by $\mathbb{Z}$ the set of integers and $\mathbb{N} = \{0, 1, 2, \dots\}$ the set of natural numbers. The notation $[n]$ represents the set of numbers from 1 to n. For two numbers $a, b \in \mathbb{Z}$, the notation $a|b$ denotes that a divides b and $\equiv_n$ denotes the congruence relation modulo n. That is $a \equiv_n b$ iff $n|a - b$.

An alphabet Σ is a finite set of symbols. A symbol in an alphabet is also called a letter . A word over Σ is a sequence of letters from Σ. The set of all finite words over Σ is denoted by Σ^* , the set of all right infinite words is denoted by Σ^ω . Let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. For a word $w \in \Sigma^\infty$ the notation $w(i)$ denotes the i^{th} letter in w , i.e. $w = w(1)w(2)w(2)\dots$. Note that a word starts from the 1^{st} position. For a word $w \in \Sigma^\infty$, we denote by $|w|$ the length of the word (possibly infinite) and the number of occurences of the letter a in w is denoted by $|w|_a$. We denote by Σ^* the set of all words over Σ and Σ^+ the set of all non-empty words. A *language* is any subset of Σ^*.

A *regular expression* over the alphabet Σ is a formula whose atomic expressions are the symbols of the alphabet and *empty word* ϵ, and closed under alternation (denoted by $+$), concatenation (.) and star operation (*). That is

$$e ::= a \in \Sigma \mid \lambda \mid e_1 \ + \ e_2 \mid e_1.e_2 \mid e_1^*$$

A *star free expression* is a regular expression whose atomic expressions are the symbols of the alphabet, the empty set (denoted by $\emptyset$) and *empty word* ϵ, and closed under alternation (denoted by $+$), concatenation (.) and negation ($\neg$). That is

$$e ::= a \in \Sigma \mid \emptyset \mid \lambda \mid e_1 \ + \ e_2 \mid e_1.e_2 \mid \neg\, e_1$$

Note that star free expressions can use negations (and hence intersection) and also the empty language, but not the star operation. A *regular language* (*star free language*) is a

language definable by a regular expression (star free expression). The book [HU79] gives an introduction to regular expressions and their connections to finite automata.

$L \subseteq \Sigma^*$ is an *unambiguous language* if and only if L is a finite union of unambiguous concatenations $K = A_0^* a_1 A_1^* a_2 ... a_t A_t^*$, with $a_i \in \Sigma$ and $A_i \subseteq \Sigma$.

2.2 COMPLEXITY CLASSES

In this thesis, we identify the computational complexity of a variety of problems. We assume the reader is familiar with complexity classes like P, NP, PSPACE etc. A short description of the not so famous classes used in the thesis follows. The class Π_2^P is an NP machine accessing an NP oracle and the class Σ_3^P is an NP machine which access a Π_2^P oracle.

We also look at certain circuit classes. The circuit family class AC^0 is defined as the family of languages recognized by constant depth polynomial sized family of *circuits* having unbounded fan-in *AND*, and *OR* gates. Similarly $ACC^0(p)$ is the family of languages recognized by constant depth polynomial sized family of circuits containing unbounded fan-in *AND*, *OR* and MOD_p for $p > 0$. Similarly $CC^0(p)$ corresponds to constant depth, polynomial size circuits with only MOD_p gates. $ACC^0(CC^0)$ is defined as the set of languages recognized by an $ACC^0(p)$ ($CC^0(p)$) family of circuits for some $p > 0$. The circuit class TC^0 corresponds to circuits with constant depth, polynomial size and having in addition to *AND* and *OR* gates *MAJ* (majority) gate. On the other hand NC^1 circuits are defined polynomial sized, log depth circuits containing *AND* and *OR* gates but having constant fan-in. There is an alternate characterization for NC^1. It is the family of languages recognized by constant depth, polynomial sized family of circuits which uses *AND*, *OR* and finite group gates. The reader can refer to the books [Vol99], [Juk12] [AB09] to know more about these classes.

Results by Razborov [Raz89] and Smolensky [Smo87] shows that:

Theorem 2.2.1. *[Raz89, Smo87] If p is a prime number and q is a prime other than p then the language $\mathcal{L}_q$ is not contained in $ACC^0(p)$.*

Hence we can infer the following: AC^0 is separated from $ACC^0(p)$ for a $p > 0$ [FSS84]; there are languages in $CC^0(p)$ which are not in AC^0; the classes $ACC^0(p)$ and $ACC^0(q)$ are different from each other if p and q are distinct primes. But relationships between most other classes are open. For example, we do not know whether CC^0 is different from ACC^0. In fact we do not know whether $CC^0(6)$ contains AC^0 or whether $CC^0(6)$ is even distinct from NP. These are among the biggest unsolved problems in circuit complexity.

We say that a family of circuits $(C_1, C_2, \dots)$ is *dlogtime-uniform* if there is a deterministic Turing machine running in logarithmic space which outputs the circuit C_n on an input 1^n. That is the n^{th} circuit in a *dlogtime-uniform*-AC^0 circuit family can be outputted

by a deterministic log space machine which takes an n bit string of all 1s. Complexity theoreticians consider this as a "natural" definition of uniformity.

A *linear bounded automata* (LBA in short) is a restricted Turing machine which can use only space which is linear function of the input size. There is no such space restriction for a general Turing machine. A *deterministic LBA* further restricts the LBA to have only deterministic moves.

2.3 SEMIGROUPS

A semigroup is a set closed under a binary associative operation (books [How95, Pin86] study semigroups). If in addition it has an identity element , then it is called a monoid . For a monoid M, we denote by 1_M the identity element of M. All monoids we consider except for Σ^* will be finite. For a finite monoid, M and an arbitrary element $m \in M$ it is known that $m^{|M|}$ is 1_M. That is m multiplied $|M|$ times will give the identity element.

Proposition 2.3.1. *Let M be a finite monoid and $m \in M$. Then $m^{|M|} = 1_M$.*

A monoid M and a subset S such that $S \subseteq M$ define a *word problem*. It is composed of words $w \in M^*$, such that when the elements of w are multiplied in order we get an element in S. That is, it is the language

$$\{w \in M^* \mid \prod_{i=1}^{|w|} w(i) \in S\}$$

We denote by U_1 the monoid consisting of elements $\{0,1\}$ under the operation $0.1 = 1.0 = 0.0 = 1$ and $1.1 = 1$. We say that a monoid M *divides* a monoid N if there exists a submonoid N' of N and a surjective morphism from N' to M. A monoid M *recognizes* a language $L \subseteq \Sigma^*$ if there exists a morphism $h : \Sigma^* \to M$ and a subset $T \subseteq M$ such that $L = h^{-1}(T)$. It is known that finite monoids recognize exactly the set of regular languages [Myh57]. We denote by $\mathcal{M}$ the set of all finite monoids. For a language L, the *syntactic monoid* is the smallest monoid that recognizes L.

Example 2.3.2. *Let $L \subseteq \{a,b\}^*$ be the set of all words containing an even number of a's. The two cycle group, $G = \{1,g\}$ is the smallest monoid which recognize the language. Consider the following morphism, $h(a) = g$ and $h(b) = 1$. Then $h^{-1}(1)$ is exactly the language L.*

Here G is the syntactic monoid *for L. Also the group G* recognizes *the language L.*

It is known that the syntactic monoid of a regular language is unique.

We also use the *block product* of monoids, defined in [RT89]. The definition can also be found in [Str94].

Definition 2.3.3. *Let M and N be two finite monoids. Let $\mathcal{F}$ be the set of all functions from $N \times N$ to M. Then the block product $M \square N$ is a monoid whose elements are $M^{N \times N} \times N$ and whose operation is given as follows (we use multiplication as the symbol of operation for readability):*

$$(F, n)(F', n') = (G, nn')$$

where $F, F', G \in \mathcal{F}$ and for all $(m, m') \in N \times N$,

$$G(m, m') = F(m, m'n').F'(mn, n')$$

For a set of monoids T, we denote by $bpc(T)$ the set got by closing T under block product operation.

A group is a monoid, with the additional property that every element has an inverse element . We denote by $\mathcal{G} \subset \mathcal{M}$ the set of all finite groups. A *symmetric group* S_n on a finite set of n symbols is the group whose elements are all the permutation of n symbols and the group operation being composition. A generating set of a group G is a subset of the elements in G which when closed under the group operation gives G. All symmetric groups has a generating set of cardinality two. A group with a generating set of cardinality one (generator) is called a cyclic group . We denote by $\textsc{mod} \subset \mathcal{G}$ the set of all finite cyclic groups. All monoids in $bpc(\textsc{mod})$ are called solvable groups and all other groups are called non-solvable groups . Solvable monoids are monoids which does not have any non-solvable subgroup. Aperiodic monoids are those monoids in $bpc(U_1)$. The variety **DA** [TT02] is an important class of monoids strictly contained inside aperiodic monoids. They have nice algebraic characterizations. We give here an equivalent definition. **DA** consists of the set of all languages recognizable by a partially ordered two way DFA (a DFA is partially ordered then all its strongly connected components are trivial).

The following decomposition theorem helps one to understand monoids better [KR65] [1]

Theorem 2.3.4. *Krohn-Rhodes decomposition theorem [KR65]*
Every finite monoid M divides a block product $N_1 \square (N_2 \square (\dots (N_{k-1} \square N_k) \dots))$, where for all $i \leq k$, N_i is either U_1 or a cyclic group or a non-solvable group.

Let us look at some regular languages.

Example 2.3.5.

- *Let $L \subseteq \{a, b\}^*$ be the set of all words of even length. The syntactic monoid of this language is not aperiodic.*
- *Let $L \subseteq \{a, b\}^*$ be the set of all finite words where the number of occurrence of letter a is $0 \ (mod\ 3)$. L is recognized by a cyclic group of length 3. That is*

$$L = \{w \mid |w|_a \equiv 0 \ (mod\ 3)\}$$

[1] The original Krohn-Rhodes theorem was stated in terms of wreath products of semigroups. We state the result in terms of block products. Straubing's book [Str94] gives the version we use.

2.4 LINEAR TEMPORAL LOGIC

Linear temporal logic [Pnu77b] is a logic where we have propositions qualified in terms of time. A *linear temporal logic* formula over a set of propositions $\mathcal{P}$ is built using the following syntax

$$\phi ::= p \in \mathcal{P} \mid \neg\, \phi \mid \phi_1 \ \vee\ \phi_2 \mid \mathrm{X}\phi \mid \mathrm{Y}\phi \mid \mathrm{F}\,\phi \mid \mathrm{P}\,\phi \mid \phi_1\ \mathrm{U}\ \phi_2 \mid \phi_1\ \mathrm{S}\ \phi_2$$

Linear temporal logic (LTL) formulas are interpreted on strings over the alphabet $\Sigma = 2^{\mathcal{P}}$, the set of all subsets of $\mathcal{P}$. We consider only finite words.

Given a finite string $u \in \Sigma^+$, a position $i \in \mathbb{N}$ such that $1 \leq i \leq |u|$ and a linear temporal logic formula ϕ over $\mathcal{P}$, we denote by $(u,i) \models \phi$ that ϕ is true at position i in the word u. The *semantics* of the logic is given below

(u,i)	$\models$	p	if $p \in u(i)$ and $p \in \mathcal{P}$
(u,i)	$\models$	$\neg\phi$	if not $(u,i) \models \phi$
(u,i)	$\models$	$\phi_1 \vee \phi_2$	if $(u,i) \models \phi_1$ or $(u,i) \models \phi_2$
(u,i)	$\models$	$\mathrm{X}\phi$	if $i < \lvert u\rvert - 1$ and $(u,i+1) \models \phi$
(u,i)	$\models$	$\mathrm{Y}\phi$	if $i > 1$ and $(u,i-1) \models \phi$
(u,i)	$\models$	$\phi_1 \mathrm{U} \phi_2$	if there exists a $j \geq i$ and $(u,j) \models \phi_2$ and for all k, if $i \leq k < j$ then $(u,k) \models \phi_1$
(u,i)	$\models$	$\phi_1 \mathrm{S} \phi_2$	if there exists a j such that $1 \leq j \leq i$ and $(u,j) \models \phi_2$ and for all k, if $j < k \leq i$ then $(u,k) \models \phi_1$

We denote by *true* the statement $p \vee \neg p$, where $p \in \mathcal{P}$. Then $\mathrm{F}\phi$, $\mathrm{P}\phi$ is equivalent to $\mathit{true}\mathrm{U}\phi$ and $\mathit{true}\mathrm{S}\phi$ respectively. As usual $\mathrm{G}\phi$ abbreviates $\neg\mathrm{F}\neg\phi$ and $\mathrm{H}\phi$ abbreviates $\neg\mathrm{P}\neg\phi$. We also define modalities $\phi_1 \breve{\mathrm{U}} \phi_2, \phi_1 \breve{\mathrm{S}} \phi_2$ which are equivalent to $\mathrm{X}\,(\phi_1 \mathrm{U} \phi_2)$ and $\mathrm{Y}\,(\phi_1 \mathrm{S} \phi_2)$ respectively. In fact

$(u,i) \models \phi_1 \breve{\mathrm{U}} \phi_2$ if there exists a $j > i$ and $(u,j) \models \phi_2$ and for all k, if $i < k < j$ then $(u,k) \models \phi_1$

Note that $\mathrm{X}\phi$ is equivalent to $\mathit{false}\ \breve{\mathrm{U}}\ \phi$ and $\phi_1 \mathrm{U} \phi_2$ is equivalent to $\phi_2 \vee (\phi_1 \wedge \phi_1 \breve{\mathrm{U}} \phi_2)$. Similarly we define $\breve{\mathrm{F}} ::= \mathrm{XF}, \breve{\mathrm{P}} ::= \mathrm{YP}, \breve{\mathrm{G}} ::= \mathrm{XG}, \breve{\mathrm{H}} ::= \mathrm{YH}$.

2.4.1 *Modulo counting operators*

The *modulo counting* operators for LTL were introduced by Baziramwabo, McKenzie and Thérien [BMT99]. The syntax of the operator, for all $r, q \in \mathbb{N}$ such that $q > 1$ and $0 \leq r < q$, is as follows.

$$\mathrm{MOD}^F_{r,q}\phi \mid \mathrm{MOD}^P_{r,q}\phi$$

The semantics for these operators are given as follows.

$$u, i \models \mathrm{MOD}^F_{r,q}\phi \text{ iff } |\{i \leq l \leq |u| \mid u, l \models \phi\}| \equiv_q r$$

$$u, i \models \text{MOD}^P_{r,q}\phi \text{ iff } |\{1 \leq l \leq i \mid u, l \models \phi\}| \equiv_q r$$

Observe that the P in the superscript denotes a past operator, whereas the F in the superscript denotes a future operator. Also note that the counting starts from the current state.

We denote by $\text{MOD}^F(q)$ the set of all operators $\text{MOD}^F_{r,q}$, where $0 \leq r < q$. Similarly we define $\text{MOD}^P(q)$ and $\text{MOD}(q)$ which is the union of both past and future such operators. That is

$$\text{MOD}^F(q) = \bigcup_{0 \leq r}^{q-1} \text{MOD}^F_{r,q}$$

$$\text{MOD}^P(q) = \bigcup_{0 \leq r}^{q-1} \text{MOD}^P_{r,q}$$

$$\text{MOD}(q) = \text{MOD}^F(q) \cup \text{MOD}^P(q)$$

Similarly we denote by MOD^F and MOD^P the set of all operators $\text{MOD}^F(q)$ for all $q > 1$ and the set of all operators $\text{MOD}^P(q)$ for all $q > 1$ respectively. We also define MOD the set of all modulo counting operators.

$$\text{MOD}^F = \bigcup_{q>1} \text{MOD}^F(q)$$

$$\text{MOD}^P = \bigcup_{q>1} \text{MOD}^P(q)$$

$$\text{MOD} = \text{MOD}^F \bigcup \text{MOD}^P$$

We now introduce operators which are expressively less powerful than the modulo counting operators we saw above. Let $r, q \in \mathbb{N}$ such that $q > 1$ and $0 \leq r < q$. Then the following are called *length counting operators*.

$$\ell^F_{r,q} \mid \ell^P_{r,q}$$

The semantics of the formula $\ell^F_{r,q}$ is equivalent to $\text{MOD}^F_{r,q}true$. That is

$$u, i \models \ell^F_{r,q} \text{ iff } u, i \models \text{MOD}^F_{r,q}true$$

$$u, i \models \ell^P_{r,q} \text{ iff } u, i \models \text{MOD}^P_{r,q}true$$

Using these operators we define the following class of operators.

$$\text{LEN}^F(q) = \bigcup_{0 \leq r}^{q-1} \ell^F_{r,q}$$

$$\text{LEN}^P(q) = \bigcup_{0 \leq r}^{q-1} \ell^P_{r,q}$$

$$\text{LEN}^F = \bigcup_{q>1} \text{LEN}^F(q)$$

$$\text{LEN}^P = \bigcup_{q>1} \text{LEN}^P(q)$$

$$\text{LEN} = \text{LEN}^F \bigcup \text{LEN}^P$$

Note that such formulas can count lengths rather than the number of occurrences of propositions or formulae.

We also look at the operator DUR, which consists of the set of all operators of the form:

$$\text{MOD}^P_{r,q} p \quad \text{where } p \in \mathcal{P}, \text{ the set of propositions}$$

That is the operator can only count the number of occurences of a proposition.

At times we may need to distinguish between the syntax where r and q in the above operators are given in binary notation and when they are given in unary.

2.4.2 *Group operators*

Now we follow Baziramwabo, McKenzie and Thérien [BMT99] to generalize the modulo counting to finite group operators. The syntax of these operators are as follows:

$$G^F_g\langle\phi_1,\ldots,\phi_k\rangle \mid G^P_g\langle\phi_1,\ldots,\phi_k\rangle$$

Here G is a finite group whose elements are $\{g_1, ..., g_k, 1\}$ and g is an element in G. Also 1 denotes the identity of G. Note that the syntax also needs to specify the group multiplication table and also the ordering of the group elements $g_1,\ldots,g_k$, but we choose to ignore it for readability. For an ordered set of formulas $\Phi = \langle\phi_1,\ldots,\phi_k\rangle$ and a word $u \in \left(2^{\mathcal{P}}\right)^*$ and an $l \in \mathbb{N}$, such that $1 \le l \le |u|$ we define $\Gamma^\Phi(u,l) \in G$ as follows:

$$\Gamma^\Phi(u,l) = \begin{cases} g_1 & \text{if } w,l \models \phi_1 \\ g_2 & \text{if } w,l \models \neg\phi_1 \wedge \phi_2 \\ \quad\vdots & \\ g_K & \text{if } w,l \models \neg\phi_1 \wedge \cdots \wedge \neg\phi_{K-1} \wedge \phi_K \\ 1 & \text{otherwise} \end{cases}$$

If Φ is obvious from the context we use $\Gamma(u,l)$. Then

$$u,i \models G^F_g\langle\phi_1,\ldots,\phi_k\rangle \text{ iff } \prod_{l=i}^{|u|} \Gamma(u,l) = g$$

$$u,i \models G^P_g\langle\phi_1,\ldots,\phi_k\rangle \text{ iff } \prod_{l=1}^{i} \Gamma(u,l) = g$$

For a $j \leq K$, we denote by Γ_j^{Φ} the formula $\neg\phi_1 \wedge \cdots \wedge \neg\phi_{j-1} \wedge \phi_j$. Again we drop the superscript Φ when the context is obvious.

For a finite group G, we denote by $\text{GRP}(G)$ the set of all group operators using the group G. We denote by GRP the class of all group operators for all finite groups G. That is

$$\text{GRP} = \bigcup_{G \in \mathcal{G}} \text{GRP}(G)$$

Observe that this is a generalization of the modulo counting we did earlier, since modulo counting is similar to working with cyclic groups. For example. $\text{MOD}_{1,q}^{F}\phi$ can be expressed by the following formula which uses the cyclic group, $C_q = \{g_1, g_1^2, g_1^3, \ldots, g_1^q\}$.

$$G_{g_1}^{F}\langle \phi, false, \ldots, false \rangle$$

Succinct representation of groups

There is another way to represent the groups. And that is to use *permutation groups* (subgroups of symmetric groups). For instance, we could specify the symmetric group S_5 (shown in Figure 1) using a syntax such as

group S_5 generators $(23451), (21345)$

which specifies a permutation group named S_5 with two generators defined as permutations of the elements $(1, 2, 3, 4, 5)$ mapping these elements to the values shown. That is the notation (23154) denotes the permutation $1 \mapsto 2, 2 \mapsto 3, 3 \mapsto 1, 4 \mapsto 5, 5 \mapsto 4$. Moreover rather than have formulas for the entire group, we specify only a generator of the group and have as many formulas as the number of generators. That is, the size of a group quantifier will depend on the number of generators of the group, rather than the size of the group itself. This is a succinct way to represent groups.

In general we define a group named G with permutations over the set $\{1, \ldots, n\}, n \geq 2$ and generators $S = \{g_1, \ldots, g_k\}$. Again the syntax will look the same.

$$G_g^F\langle \phi_1, \ldots, \phi_k \rangle \mid G_g^P\langle \phi_1, \ldots, \phi_k \rangle$$

The only difference being how the groups are now encoded and that $\Gamma(u, l)$ are mapped to the generators rather than group elements. Notice that g in the syntax above is a group element, not necessarily a generator of the group. The advantages of this notation are

- Every group is isomorphic to a subgroup of a symmetric group [Her64].
- One can now specify the group elements as a permutation. Moreover the group multiplication is now implicit and thus the group multiplication table need not be given along with the formula. This way of representing groups will reduce the size of the formulas.

- Using the generators is also a succinct way of representing groups. For instance, the symmetric group S_n has $n!$ elements, but can be generated by 2 generators (as shown in the above example) each generator being a permutation on n elements. In general, though any group has a generating set of logarithmic size.

  Proposition 2.4.1. *[vzGG03] Any group has a generating set of logarithmic size.*

  Proof. Let G be a group. For an $H \subseteq G$, we denote by $\langle H \rangle$ the group generated by the elements H. Take an element $g_0 \in G$. Let $H_0 = \{g_0\}$. If $\langle H_0 \rangle \neq G$, take g_1 from $G \backslash \langle H_0 \rangle$, and call $H_0 \cup \{g_1\}$ as H_1. Continue doing this until you find an H_k such that $\langle H_k \rangle = G$. We prove that $\forall i \leq k : |\langle H_{i+1} \rangle| \geq 2 \times |\langle H_i \rangle|$. Observe that since $g_{i+1} \notin \langle H_i \rangle$, it implies $g_{i+1}\langle H_i \rangle \cap \langle H_i \rangle = \emptyset$. Also $|g_{i+1}.\langle H_i \rangle| = |\langle H_i \rangle|$. But $g_{i+1}.\langle H_i \rangle \cup \langle H_i \rangle \subseteq H_{i+1}$. Therefore $|\langle H_{i+1} \rangle| \geq 2 \times |\langle H_i \rangle|$. Hence $\langle H_{log|G|} \rangle = G$. □

- There are permutations on n elements whose order is exponential in n and thus to represent such groups we only need to use logarithmic many bits in our input representation.

  For example, let $p_1, ..., p_n$ be the first n prime numbers and $\forall i \leq n$, let s_i be defined as $\sum_{j=1}^{i} p_j$. We claim that the cyclic group (given in cyclic form) generated by $(1, ..., s_1)(s_1 + 1, ..., s_2)....(s_{n-1} + 1, ..., s_n)$ is of size $\Pi_{i=1}^{n} p_i$. This follows from the well known fact.

  Proposition 2.4.2. *[Her75] Order of a permutation given in cyclic form, is the lcm of the length of its cycles.*

  Thus to represent the above group of size $\prod_{i=1}^{n} p_i$, which is exponential in n, we require space $\sum_{i=1}^{n} p_i \leq n^2$, which is polynomial in n.

  Observe that the other direction need not hold. That is, there are groups which cannot be represented as permutations on sets of size less than the group.

  For example, the proposition 2.4.2 gives us that, no generator of a cyclic group of size 2^n can be represented by a permutation on a set of size less than 2^n.

Consider the following example. Figure 1 shows the symmetric group S_n (for $n = 5$) as the transition structure of an automaton. The language accepted can be defined by the formula

$$G^{F}_{(12...n)}\langle a, b \rangle$$

where G is the group S_5 shown earlier and $(12 \ldots n)$ is the identity permutation of the group.

2.4.3 *Extended Temporal Logics*

We denote by O the set of all the temporal operators we saw. That is

$$O = \{\mathrm{X}, \mathrm{Y}, \mathrm{S}, \mathrm{U}, \mathrm{F}, \mathrm{P}, \textsc{mod}, \textsc{mod}(q), \textsc{dur}, \textsc{len}, \textsc{grp}, \textsc{grp}(G)\}$$

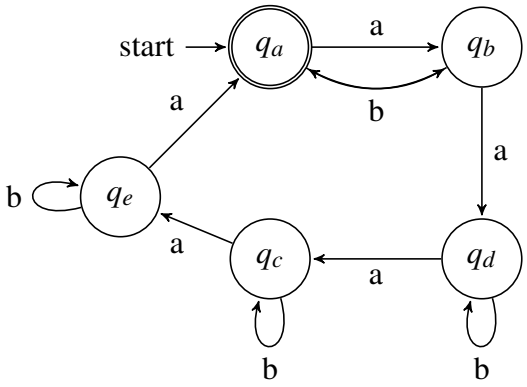

Figure 1: An automaton representing the symmetric group S_5

Let $S \subseteq O$. Then, we denote by TL[S], the logic formed by closing propositions $\mathcal{P}$ with disjunctions, negations, and modal operators from S. For example, the logic TL[U, X, S, Y] denotes LTL. The logic TL[$\emptyset$] denotes propositional logic. The logic TL[X, Y, F, P] denotes UTL. We denote by LTLGRP

$$\text{LTLGRP} = \text{TL}[\text{X}, \text{Y}, \text{S}, \text{U}, \text{F}, \text{P}, \text{MOD}, \text{DUR}, \text{LEN}, \text{GRP}]$$

We also denote by LTLGRP(G) the logic

$$\text{LTLGRP(G)} = \text{TL}[\text{X}, \text{Y}, \text{S}, \text{U}, \text{F}, \text{P}, \text{GRP}(G)]$$

Similarly LTLMOD stands for TL[X,Y,S,U,MOD].

Recall that the groups can be represented in "succinct" notation or not, where the succinct representation of groups are by viewing them as subgroups of symmetric groups and then using only generators of the groups. When we represent modulo operators we have the liberty to represent the numbers in binary or unary notation. We denote by LTLGRP^{bin} for TL[O] if the groups are represented succinctly and the integers in the modulo counting operators are represented in binary notation. On the other hand we use LTLGRP^{un} to denote the same logic, where groups are represented by its multiplication table and the modulo counting operators are represented in unary notation.

Example 2.4.3. *Consider the language $L' = \{w \in \{a,b\}^* \mid |w|_a$ is even$\}$. Again this language is not expressible in* LTL. *The following formula in* TL[F,MOD] *expresses L'.*

$$\text{MOD}^F_{0,2} a$$

Example 2.4.4. *Consider a language $L \subseteq \{a,b\}^*$ such that the number of times b occurs such that there are an even number of as to the right of it is* 1 $(mod\ 3)$. *That is*

$$L = \{w \mid |\{i \mid w[i] = b \text{ and } w[i+1,|w|] \text{ contains even number of } a\}| \equiv 1 \ (mod\ 3)\}$$

The following formula uses the expression for the language L' seen in the previous example, to describe L.

$$\text{MOD}^F_{1,3}\Big(b \wedge \text{MOD}^F_{0,2} a\Big)$$

Example 2.4.5. *Let $L \subseteq \{a,b\}^*$ be the set of all words of even length. We know that L is not an aperiodic language (see Straubing's book [Str94]). We show that the formula is definable in* TL*[*F,X,LEN*].*

The statement "$\neg(\mathrm{X}\ true)$" is true only in the last state of a word. If the last state is an even position then it satisfies the formula, "$\ell^P_{0,2}$". Combining the conditions gives us a formula which recognizes L.

$$\mathrm{F}\Big((\neg \mathrm{X}\ true) \wedge \ell^P_{0,2}\Big)$$

2.4.4 *Other definitions*

For a formula $\phi \in$ LTLGRP we say that ϕ *satisfies* a word u if $(u,1) \models \phi$. Then u is called a *model* of ϕ. We denote by $L(\phi)$ the language of ϕ, that is the set of all the models of ϕ. We say that formulas α and β are *equivalent* if for all words u and for all i such that $1 \leq i \leq |u|$, we have $u,i \models \alpha \Leftrightarrow u,i \models \beta$. We say that formulas α, α' are expressively equivalent if $L(\alpha) = L(\alpha')$.

Let $S \subseteq O$. The *satisfiability problem* for a logic TL$[S]$ takes as input a formula $\phi \in$ TL$[S]$. The problem is to check whether there exists a word u, such that u is a model of ϕ.

Let $\mathcal{P}$ be a set of propositions. A *Kripke structure* $K = (S, R, L, s_0)$ is a rooted *transition system* with the following properties: S is a finite set of states, a transition relation $R \subseteq S \times S$, a labelling function $L : S \to \mathcal{P}$ and an initial state, $s_0 \in S$. The *model checking problem* for a Kripke structure and a formula α checks whether all runs of the transition system are models of α.

We say that an LTLGRP formula is a *pure future formula* if the only modalities used are future operators. Similarly we say that an LTLGRP formula is a *pure past formula* if the only modalities used are past operators. *Pure present formulas* are those which do not use any modality. The formula "F $\mathrm{MOD}^P_{r,q}\alpha$" is neither a pure past or a pure present or a pure future formula. In such a case we call the formula *impure* . We say that a formula can be *separated* if it can be written as a boolean combination of pure past, pure present and pure future formulas. A logic satisfies the *separation* property [Gab87] if all formulas in that logic can be separated.

We define the *future depth* (*past depth*) of a formula inductively. All pure past (future) formula has future (past) depth 0. Future depth of the formulas $\phi_1 \mathrm{U} \phi_2$, $\mathrm{X}\phi_1$, $G^F_g\langle \phi_1, \ldots, \phi_k\rangle$ is one more than the maximum of the future depth of the formulas $\phi_1, ..., \phi_k$. Similarly the past depth of the formulas $\phi_1 \mathrm{S} \phi_2$, $\mathrm{Y}\phi_1$, $G^P_g\langle \phi_1, \ldots, \phi_k\rangle$ is one more than the past depth of the formulas $\phi_1, ..., \phi_k$. The future (past) depth of $\phi_1 \vee \phi_2$, $\phi_1 \wedge \phi_2$, $\neg\phi$ is the maximum of the future depth of ϕ_1 or ϕ_2.
The *operator depth* or *depth* of a formula is the sum of its future depth and past depth. Similarly the *alternation depth* of a formula is the number of alternations of its future and past modalities.

2.5 FIRST ORDER LOGIC

Let Σ be a finite alphabet and $\mathcal{V} = \{x_1, \dots\}$ be a set of variables. Let us introduce the syntax of first order logic over word models. The signature τ consists of:

- Unary predicates a, such that $a \in \Sigma$.
- Binary relation $<$, which is a *linear order*.
- Binary relation *succ*, which is the *successor relation*, which respects the linear order.
- Binary *congruence relations* $\equiv_q$ for all $q \in \mathbb{N}$ such that $q > 1$. Let t_1 and t_2 be two terms. Then, we say that $t_1 \equiv_q t_2$ iff $t_1 - t_2$ is divisible by q. We might also have unary congruence relations , $\equiv_q r$, for all $q, r \in \mathbb{N}$ such that $q > 1$ and $0 \leq r < q$. For a term t we say $t \equiv_q r$ iff $t - r$ is divisible by q. Let $\equiv$ denote the set of all unary congruence relations.
- Ternary relation $+$, which is the *addition* relation. Sometimes, we use addition as a function. If not mentioned, always consider addition as a relation.
- Binary relations $y = nx$, where $n \in \mathbb{N}$. Note that addition can simulate this relation.
- Constants $\{0, 1, \dots\}$

We denote this logic FO[τ].

The τ-*terms* are defined inductively as follows. Variables and constants are τ-terms.

$$t ::= x \in \mathcal{V} \mid c \in \{0, 1\}$$

Atomic formulas are inductively defined as follows. If t_1, t_2, t_3 are atomic formulas, then

$$t \text{ is a } \tau\text{-term } \mid t_1 = t_2 \mid t_1 < t_2 \mid succ(t_1, t_2) \mid t_1 \equiv_q t_2 \mid t_1 = t_2 + t_3$$

are also atomic formulas. Note that equality is part of our logic.
First order formulas are now defined inductively as follows. All atomic formulas are first order formulas. If α and β are first order formulas, then the following are also first order formulas.

$$\neg\, \alpha \mid \alpha \ \vee\ \beta \mid \alpha \wedge \beta \mid \alpha \Rightarrow \beta \mid \alpha \Leftrightarrow \beta \mid \exists x \alpha \mid \forall x \alpha$$

The implication ($\Rightarrow$), if and only if ($\Leftrightarrow$), and conjunction ($\wedge$) connectives can be simulated using disjunction ($\vee$) and negation ($\neg$) connective as follows.

$$\alpha \wedge \beta = \neg(\neg\alpha \vee \neg\beta), \alpha \Rightarrow \beta = \neg\alpha \vee \beta, \alpha \Leftrightarrow \beta = (\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$$

Similarly the for all ($\forall$) quantifier can be simulated using the existential quantifier as follows:

$$\forall x \alpha = \neg \exists x \neg \alpha$$

The *free variables* of a formula is the set of variables which are not quantified. We say that a formula is a *sentence* if it does not have any free variables. A formula is called *quantifier free* if there are no quantifiers in the formula.

Now we come to the semantics of first order logic. Consider the formula α. Let w be a word and $\mathcal{I}$ be an *interpretation* which associates each numerical relation of arity k to a subset of $\{1,\ldots,|w|\}^k$. The free variables are assigned some number in between 1 and $|w|$. Now the semantics is given inductively. $w,\mathcal{I} \models a(x)$, if and only if the interpretation for x in $\mathcal{I}$ is $\mathcal{I}(l) \in [|w|]$ and $w(\mathcal{I}(l)) = a$. Similarly for a numerical relation $R(x_1,\ldots,x_k)$, we have that $w,\mathcal{I} \models R(x_1,\ldots,x_k)$ if and only if x_i is assigned $l_i \in [|w|]$ for all $i \leq k$ and $(l_1,\ldots,l_k)$ is a tuple in the interpretation for R in $\mathcal{I}$. The semantics for conjunction, disjunction and negation are as usual. We say that $w,\mathcal{I} \models \exists x\alpha$ iff there exists an interpretation, $\mathcal{I}'$ which extends $\mathcal{I}$ with an assignment for x, and we have that $w,\mathcal{I}' \models \alpha$. For an interpretation $\mathcal{I}$ we denote by $\mathcal{I}[x \mapsto i]$ the interpretation which extends $\mathcal{I}$ with the additional variable x being assigned the number $i \in \mathbb{N}$. Thus

$$w,\mathcal{I} \models \exists x\alpha \Leftrightarrow \text{there exists an } i \leq n \text{ such that } w,\mathcal{I}[x \mapsto i] \models \alpha$$

In the thesis, we will be looking only at relations less than ($<$) and addition ($+$). Therefore the interpretations for the relations are obvious and hence we drop them from the notation. Hence our interpretations will contain only assignments to the variables. We also use the notation $w,\mathcal{I} \not\models \alpha$, if w with the interpretation $\mathcal{I}$ does not satisfy α.

Example 2.5.1. $ababaa, x = 1, y = 4 \models x < y \wedge a(x) \wedge b(x)$
$ababaa, x = 3, y = 2 \not\models x < y \wedge a(x) \wedge b(x)$

The following theorem [Kam68, Gab87] connects LTLwith first order logic over words.

Theorem 2.5.2. *[Kam68, Gab87]* LTL *is expressively complete for FO[<]*

The theorem states that for all first order logic formula $\phi(x)$ with one free variable, there exists a formula ψ in LTL such that for all words u and $\forall i$ where $1 \leq i \leq |u|$ we have that

$$u,i \models \phi \Leftrightarrow u,i \models \psi$$

2.5.1 *Counting quantifiers*

We now introduce the syntax for counting capabilities. First we introduce *unary counting quantifiers* [GOR99, Ruh99a, Sch05]

$$\exists^{\sim y} x\alpha$$

Here α is inductively defined and $x,y \in \mathcal{V}$ the set of variables and $\sim \in \{<,=,>\}$. The semantics is given as follows. Let w be a word over the alphabet of α. Then

$$w,\mathcal{I} \models \exists^{\sim y} x\alpha \Leftrightarrow |\{l \mid w,\mathcal{I}[x \mapsto l] \models \alpha\}| \sim \mathcal{I}(y)$$

The *majority quantifier*, Maj $x\ \phi(x)$ is given as follows.

$$w \models \text{Maj}\ x\ \phi(x) \Leftrightarrow |\{i \mid w \models \phi(i),\ i \leq |w|\}| > \frac{|w|}{2}$$

The unary counting quantifiers and majority quantifiers are not regular quantifiers, since they can define non-regular languages even if the only relation present is equality.

In the more restricted case, we have modulo counting quantifiers, introduced by Straubing, Thérien and Thomas [STT95]. Here counting terms cannot be compared with variables, but they can be compared only with a constant modulo a number. Here is the syntax for modulo counting quantifiers.

$$\exists^{(r,q)} x(\alpha)$$

The semantics is given as follows.

$$w, \mathcal{I} \models \exists^{(r,q)} x(\alpha) \Leftrightarrow |\{l \mid \mathcal{I}, s[x \mapsto l] \models \alpha\}| \equiv r \mod q$$

Note that both the above quantifiers can simulate binary (and unary) congruence relations.

2.5.2 *Monoid/Group quantifiers*

Now we follow Barrington, Immerman and Straubing [BIS90] to generalize the modulo counting quantifiers to monoid quantifiers. We view monoid quantifiers as a special case of Lindström quantifiers [Lin66]. The formal definition of a *monoid quantifier* [BIS90] is as follows. Let $M = \{m_1, \ldots, m_K, 1\}$ be a monoid with $K+1$ elements. For an $m \in M$, the quantifier Q_M^m is applied on K formulas. Let x be a free variable and $\phi_1(x), \ldots, \phi_K(x)$ be K formulas. Consider the word $u \in M^*$, where the i^{th} letter of u is given as follows. Let $1 \leq i \leq |w|$.

$$u(i) = \begin{cases} m_1 & \text{if } w, i \models \phi_1 \\ m_2 & \text{if } w, i \models \neg\phi_1 \wedge \phi_2 \\ \vdots & \\ m_K & \text{if } w, i \models \neg\phi_1 \wedge \cdots \wedge \neg\phi_{K-1} \wedge \phi_K \\ 1 & \text{otherwise} \end{cases}$$

Then

$$w \models Q_M^m x\langle\phi_1(x), \ldots, \phi_K(x)\rangle \Leftrightarrow \prod_{i=1}^{|w|} u(i) = m$$

This generalizes the modulo counting we were doing earlier, which can be thought of as working with cyclic groups. The set of all monoid quantifiers where the monoids are groups are called *group quantifiers*

2.5.3 *Extended first order logic*

Earlier we defined $\tau = \{a, a \in \Sigma, <, +, \equiv, succ\}$. Let $\upsilon \subseteq \tau$ be a signature, such that υ contains unary predicates a, where $a \in \Sigma$. Then we denote by FO$[\upsilon]$, first order logic using the relations and constants in υ. We denote by FO$_{\text{UN}}$C$[\upsilon]$ the logic extending first order logic with unary counting quantifiers.

MAJ$[<]$ denotes the logic closed under majority quantifiers. It is known that the majority quantifiers are equivalent to unary counting quantifiers [Ruh99a]. A detailed study of majority quantifiers can be found in the thesis of Krebs [Kre08].

Closing FO$[\upsilon]$ under modulo counting quantifiers give us FOMOD$[\upsilon]$. Let t be an atomic formula in FO$[\upsilon]$. Then the logic FOMOD$[\upsilon]$ is defined as

$$\phi ::= t \mid \phi_1 \vee \phi_2 \mid \neg\phi \mid \exists x\phi \mid \exists^{r,q} x\phi, \text{ for } q > 0, 0 \leq r < q$$

If the only modulo counting quantifiers used are of the form $\exists^{r,q}, r < q$ for a fixed q, then the logic is called FOMOD$(q)[\upsilon]$. Extending FO$[\upsilon]$ with group quantifiers, give us the logic FOGRP$[\upsilon]$. If the only group we use is G, then we get the logic FOGRP$(G)[\upsilon]$. Similarly if the only quantifiers used are group quantifiers then the logic will be denoted by GROUP$[\upsilon]$.

Let $\mathcal{S}$ be a set of monoids. Then, we define the logic $\mathcal{L}_{\mathcal{S}}[\upsilon]$ to be built from the, the binary predicate $\{=\}$, the predicates in υ, the variable symbols $\mathcal{V}$, the Boolean connectives $\{\neg, \vee, \wedge\}$, and the monoid quantifiers Q_M^m, where $M \in \mathcal{S}$ is a monoid and $m \in M$. We also identify the logic class $\mathcal{L}_{\mathcal{S}}[\upsilon]$ with the set of all languages definable in it.

The following "shorthand" notation is used to avoid clutter. We denote by $Q_M^m x\, \phi\, \langle \alpha_1, \ldots, \alpha_K \rangle$, the formula $Q_M^m x \langle \phi \wedge \alpha_1, \ldots, \phi \wedge \alpha_K \rangle$. Informally, this relativizes the quantifier to the positions where ϕ is true, by multiplying with the identity of M in all other places.

Consider the monoid U_1. It is easy to see that the word problem defined by U_1 and the set $\{0\}$ defines the regular language $1^*0\{0,1\}^*$. Then $Q_{U_1}^0$ is same as the existential quantifier $\exists$, since any formula $\exists x\phi$ is equivalent to $Q_{U_1}^0 x\, \langle \phi \rangle$. So the logic $\mathcal{L}_{U_1}[<]$ denotes first-order logic, FO$[<]$. Let C_q stand for the cyclic group with q elements. Then the quantifiers $Q_{C_q}^1$ corresponds to the modulo counting quantifiers we defined earlier.

The following result connects semigroups and monoid quantifiers. It gives an algebraic characterization for the logic $\mathcal{L}_{\mathcal{S}}[<]$.

Lemma 2.5.3 ([Str94]). *Let $\mathcal{S} \subseteq \mathcal{M}$ and $L \subseteq \Sigma^*$ be such that M is the smallest monoid which recognizes L. Then L is definable in $\mathcal{L}_{\mathcal{S}}[<]$ iff M divides a monoid in $bpc(S)$.*

The notation FO$^k[\upsilon]$ denotes the sublogic of FO$[\upsilon]$ which uses only k variables. The logic permits reuse of variables though.

Presburger arithmetic is the first order theory of the natural numbers with addition. That is the signature does not contain any unary alphabet. The binary relations are linear order and addition. We denote this logic as FO over $(\mathbb{N}, <, +)$.

We denote by $qd(\alpha)$, the *quantifier depth* of a formula α. We say that a formula α is *satisfiable*, if there is a word w and an interpretation which is a model for α. The *satisfiability problem* for a logic takes as input a formula in that logic and decides whether the formula is satisfiable or not. For a sentence ϕ, we define the language of ϕ, $L(\phi) = \{w \mid w \models \phi\}$.

2.5.4 *Few examples*

Let us look at a few examples of the logics we have defined above.

Example 2.5.4. *Even length words can be expressed in FO[<, ≡] by*

$$\exists max\, \forall y\, max \geq y \wedge (max \equiv 0(\mod 2))$$

It says that the variable "max" is the last position in the word and it is even.

Example 2.5.5. *On the other hand even number of letter a requires an FO*MOD*[<] formula:*

$$\exists^{(0,2)} x(a(x))$$

Example 2.5.6. *Let $L = (ab + ba)^*$. The logic $FO^2[<, succ, \equiv]$ can express L*

$$\forall x \forall y(succ(x, y) \wedge x \equiv 1(\mod 2) \Rightarrow ((a(x) \Rightarrow b(y)) \wedge (b(x) \Rightarrow a(y))$$

The formula says that if there is an a (b) at an odd position then its successor position should be a b (a).

Example 2.5.7. *The "dot depth k" language which allows at most k more a's than b's can be defined in FO^2*UNC*[<, succ]. The formula below which uses addition can be written using successor and taking $k + 1$ disjuncts:*

$$\forall x((\#y(y \leq x \wedge a(y) \leq y) \wedge (y \leq \#y(y \leq x \wedge b(y)) + k))$$

Note that we only used two variables.

Part II

REGULAR LANGUAGES: EXTENSIONS TO FIRST ORDER LOGIC WITH LINEAR ORDERING

3

SURVEY ON REGULAR LANGUAGES

3.1 EXPRESSIVENESS

Regular languages are among the most studied area in computer science. Kleene's theorem [Kle56] established that regular expressions and finite automata have the same expressive power. Results of Myhill [Myh57] and Nerode [Ner58] showed that regular languages have finite index. Eilenberg (see book [Eil76]) looked at regular languages from an algebraic point of view and finally Büchi [Bö60] gave language preserving (and also effective) translations between *monadic second order logic* and finite automata.

The most important fragment of regular languages, are the set of languages definable by first order logic (with <). This fragment, which can be characterized by temporal logics, has found considerable practical applications. On the other hand, the fragment has nice algebraic properties and other interesting characterizations. The Figure 2 and the following theorem gives some of them.

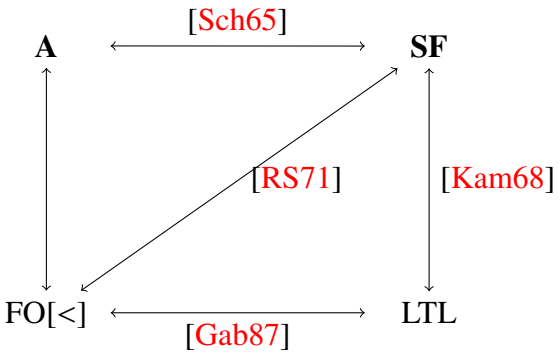

Figure 2: Star free Languages; Different characterizations

Theorem 3.1.1. *The following are equivalent.*

1. *L is definable in FO[<].*
2. *L is definable by a star free expression.*

3. *L is recognized by an aperiodic monoid (***A***).*

4. *L is definable by linear temporal logic.*

5. *L is recognized by alternating finite automata without loops.*

6. *L is definable in the three variable fragment of FO[<].*

7. *L is definable in Interval temporal logic (*ITL*).*

Schützenberger's theorem [Sch65] showed the equivalence of (2) and (3). Kamp [Kam68] then showed that LTL and star free expressions are expressively equivalent. The equivalence of (1) and (2) was shown by McNaughton and Papert [RS71]. The equivalence of (2) and (5) was shown by Salomaa and Yu [SY00]. A direct proof of the equivalence of LTL and FO[<]was given by Gabbay [Gab87] (Prior [Pri56], Pnueli [Pnu77a], Gabbay, Pnueli, Shelah and Stavi [GPSS80]). It is easy to give a translation from formulas in LTL to formulas in FO[<] with three variables, which shows that for every FO[<] formula there is an equivalent formula in FO[<] which uses only three variables. See Lodaya et al. [LPS10] for the equivalence of ITL and FO[<].

3.1.1 *Group quantifier Extensions*

We look at extending these logics with monoid (group) quantifiers. Barrington, Immerman and Straubing [BIS90] showed using the Krohn-Rhodes decomposition of finite monoids that the regular languages can also be described by FO$_{\text{GRP}}$[<], first order logic extended with group quantifiers.

Baziramwabo, McKenzie, Thérien [BMT99] use the Krohn-Rhodes decomposition (see Theorem 2.3.4) of monoids to show that LTL$_{\text{GRP}}$ formulas, LTL extended with group computation modalities, are equivalent to regular languages. This also establishes the three-variable property for FO$_{\text{GRP}}$[<]. We capture all these results in the following figure 3 and theorem 3.1.2

Theorem 3.1.2. *Let G be a finite group. The following are equivalent.*

1. *L is definable in FO*$_{\text{GRP}}$*(G).*

2. *L is definable in* LTL$_{\text{GRP}}$*(G).*

3. *L is recognized by a monoid in bpc*$\{U_1, G\}$.

We show the equivalence of 1 and 2 by proving that LTL$_{\text{GRP}}$(*G*) satisfies the seperation property [1]. The property was used by Gabbay [Gab87] to show the expressive equivalence of LTL and FO[<]. Gabbay showed that all formulas in LTL can be seperated.

[1] Seperation property was defined in Preliminaries

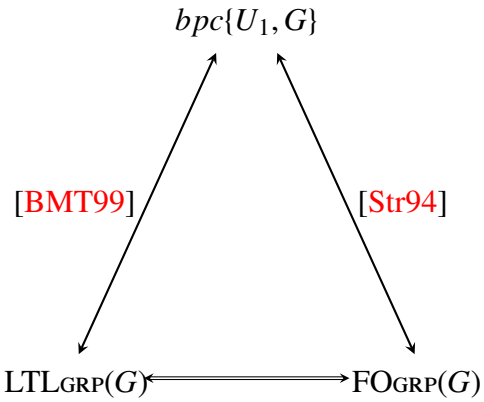

Figure 3: Group quantifiers - The double line is our contribution.

Theorem 3.1.3. *[Gab87] The logic* LTL *satisfies the separation property*

This result also shows that every LTL formula is *initially equivalent* to a formula in TL[X,U] (that is LTL without any past modalities). That is for any formula ϕ in LTL, there exists a formula ϕ' in TL[X,U] such that for all words u we have that

$$u, 1 \models \phi \Leftrightarrow u, 1 \models \phi'$$

Therefore

Theorem 3.1.4. *[Sch02], [Kam68] Every* LTL *formula is initially equivalent to an* TL[X,U] *formula.*

We give a similar result for the logic LTL$_{GRP}(G)$.

3.1.2 *A sublogic*

We saw earlier that every FO[<] formula is expressively equivalent to an FO[<] formula in three variables. The subclass of logic which uses only two variables is also interesting. As far as expressiveness is concerned it is less powerful. It is known (see [TW98] for example) that the logic FO^2[<] is a strict subset of FO^2[<, *succ*] which is a strict subset of FO[<]. The two variable fragment, FO^2[<] characterizes the class of *unambiguous languages*. This class also has other interesting characterizations as given in Figure 4 and the following theorem.

Theorem 3.1.5. *The following are equivalent.*

1. *L is definable in FO^2[<].*
2. *L is an unambiguous language.*
3. *L is recognized by a monoid in variety* **DA**.

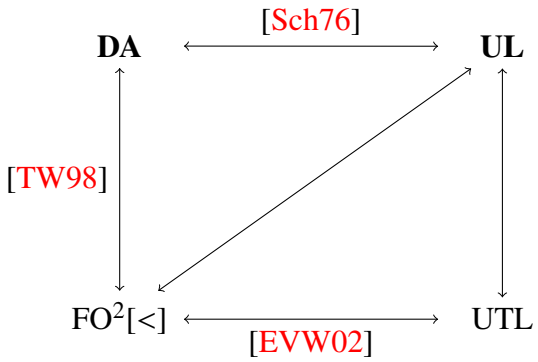

Figure 4: Unambiguous Languages; Different characterizations

4. *L is definable in* UTL.

5. *L is in* $\Sigma_2[<] \cap \Pi_2[<]$.

6. *L is definable in Unambiguous interval temporal logic (*UITL*).*

The equivalence of $(2), (3)$ and (5) were shown in [Sch76] and [PW97] and (1) and (4) where shown to be equivalent by Etessami, Vardi and Wilke [EVW02]. See Lodaya et al. [LPS10] for the equivalence of (6) and (1). Finally, the equivalence of (1) and (3) were shown in [TW98].

Straubing, Therien and Thomas show in [STT95] that every formula in FO$_{\text{MOD}}$[<] (the logic got by extending FO[<] with modulo counting quantifiers) is equivalent to a formula in FO$_{\text{MOD}}$[<] with only three variables. The authors also looked at the two variable sublogic and showed that it is a strict subset. They also show that the class is equivalent to the set of languages recognized by the variety $DA * Gsol$ [2]. Straubing and Therien [ST03] also observed the following interesting property

$$\text{FO}^2\text{MOD}[<] \cap \text{FO}[<] \not\subseteq \text{FO}^2[<]$$

Consider the language $L = (ab)^*$. It can be defined in FO^{2}MOD[<] as follows [3]:

$$\forall y\, b(y) \Leftrightarrow \exists^{(0,2)} x(x < y)$$

It states that a word is in L iff in the word the letter b is in a position if and only if it is an even positions. It is also possible to show that L is definable in FO[<] and not definable in FO^2[<].

In Chapter 4 we show the equivalence of LTL$_{\text{GRP}}$ and FO$_{\text{GRP}}$[<]. In fact we give a direct translation of an FO$_{\text{GRP}}$[<] formula to an LTL$_{\text{GRP}}$ formula. The proof goes via showing a "separation property" for LTL$_{\text{GRP}}$.

[2] *Gsol* denotes the set of all solvable groups
[3] The syntax/semantics is given in the Preliminaries chapter 2

3.2 SATISFIABILITY AND MODEL CHECKING

The satisfiability and model checking questions for first order logic has been looked at by Meyer and Stockmeyer [SM73]. From the construction of Büchi [Bö60] we know that any formula in first order logic can be translated into some automata in non-elementary time [4]. Thus the question of satisfiability of first order logic gets reduced to non-emptiness in automata. Thus we get an upper bound of non-elementary time for satisfiability of first order logic. Stockmeyer [Sto74] gives a corresponding lower bound also, which is an extension of the work of Meyer and Stockmeyer [SM73] showing that the non-emptiness of starfree regular languages is in non-elementary time.

Theorem 3.2.1. *[Bö60, Sto74, SM73] The satisfiability of FO[<] has an upper bound of non-elementary space.*
The satisfiability of FO[<] has a lower bound of non-elementary space.
[Var82] The model checking of FO[<] is PSPACE*-complete.*

The satisfiability of the two variable fragment of FO[<] has also generated considerable interest. Etessami, Vardi and Wilke [EVW02] first showed that the satisfiability of $FO^2[<]$ is NEXPTIME-complete, whereas from [Sto74] we have that the satisfiability of FO[<] is non-elementary complete. Weis and Immerman [WI09], then showed that $FO^2[<]$ is NP-complete, if the alphabet size is constant, whereas $FO^2[<, succ]$ is NEXPTIME-complete even for constant alphabet size.

Theorem 3.2.2. *[EVW02] The satisfiability of $FO^2[<]$ is* NEXPTIME*-complete for a growing alphabet.*
[EVW02] The satisfiability of $FO^2[<, succ]$ is NEXPTIME*-complete for any alphabet.*
[WI09] The satisfiability of $FO^2[<]$ is NP*-complete for a constant alphabet.*

Let us now turn to Linear temporal logic. The advantage of LTL over FO[<] is that there is a fast reduction from LTL to automata. This permits faster algorithms for checking for satisfiability of this logic. Hence LTL has found wide use as a specification language in industry. From the results of Pnueli [Pnu77c] we know the following.

Theorem 3.2.3. *[Pnu77c, SC85] The satisfiability and model checking problem for* LTL *is* PSPACE*-complete.*
The hardness holds even for the logic TL[X,F].

We can do better if X is not present.

Theorem 3.2.4. *[ON80] The satisfiability and model checking problem for* TL*[F,P] is* NP*-complete.*

[4] The height of the tower corresponds to the alternation between negations and existential quantifiers in the first order logic formula.

Serre had looked at the satisfiability of LTLMOD in [Ser04]. He reduced the problem to emptiness checking in alternating finite automata and showed a PSPACE-upper bound. But his method assumes that the "numbers" used to denote modulo counting are represented in unary notation. Hence, if the numbers are represented in binary notation, his approach gives an EXPSPACE-upper bound. We later show in Chapter 5 that this can be improved to PSPACE even for binary notation.

Theorem 3.2.5. *[Ser04] The satisfiability and model checking problem for* LTLGRP *is in* EXPSPACE.

A comparison of these results have been collected in Table 1.

	Membership for words	Emptiness/non-satisfiable
DFA	LOGSPACE-complete [JR91]	NLOGSPACE-complete
NFA	NLOGSPACE-complete [JR91]	NLOGSPACE-complete
SF	P-complete [Pet02, Pet00]	NON-ELEMENTARY-complete [SM73]
FO[<]	PSPACE-complete [Var82]	NON-ELEMENTARY-complete [Sto74]
$FO^2[<]$	P	NP-complete [WI09]
$FO^2[<, succ]$	P	NEXPTIME-complete [EVW02]
LTL	NC^1-hard,P [DS02]	PSPACE-complete [SC85]
LTLMOD	NC^1-hard,P	EXPSPACE [Ser04]

Table 1: Comparison of Satisfiability/Membership of word models for various logics (assumes constant alphabet). Note that the model checking problem for LTL is PSPACE-hard, where the given model is a Kripke structure. This table assumes that the given model is a word.

We look at the satisfiability and model checking problems for these logics when extended with modulo counting quantifiers and group quantifiers.

In Chapter 5 we look at the satisfiability and model checking for LTLGRP and its sublogics.

In Chapter 6 we look at these questions for FO[<] with modulo counting (and group quantifiers) and its various sublogics, especially the two variable fragment of this logic.

4

LTLGRP EXPRESSIVENESS (EQUIVALENCE WITH FOGRP[<])

4.1 INTRODUCTION

In this chapter we give a Gabbay-style [Gab87] separation-based proof of the equivalence of LTLGRP and FOGRP[<]. We show that LTLGRP(G) has the separation property and using that to show that the logic is equivalent to FOGRP(G)[<].

We also look at unary temporal logic UTL[EVW02]. Recall that UTL=TL[X,Y,F,P]. We extend the technique of Etessami, Vardi and Wilke ([EVW02]) to show that UTL extended with group operators has the same expressive power as FOGRP[<, *succ*] which uses only two variables.

In sections 4.2 and 4.2.1 we will be working over the strict until, $\breve{\mathrm{U}}$ and strict since, $\breve{\mathrm{S}}$ modalities. Note that, strict until (strict since) can be simulated by until (since) and next (yesterday) operator and vice versa.

Recall from the Preliminaries that for an ordered set of formulas $\langle \phi_1, \ldots, \phi_k \rangle$, the notation Γ_j denotes the formula $\phi_j \wedge \bigwedge_{i<j} \neg \phi_i$.

4.2 PROPERTIES OF LTLGRP

We first look at certain properties of the logic LTLGRP. Our first observation is that the formulas in LTLGRP have a normal form.

Theorem 4.2.1. *Let $\alpha \in$ LTLGRP. Then there exists a group G_α and a formula $\hat{\alpha} \in$ LTLGRP(G_α) such that α and $\hat{\alpha}$ are expressively equivalent. That is for all word models u and for all $i \leq |u|$*

$$u, i \models \alpha \Leftrightarrow u, i \models \hat{\alpha}$$

Proof. Take G_α to be the cross product of all the groups in α. Now any group G can be replaced by G_α by suitably choosing the accepting group element. □

Hence we can always work with formulas over LTL$_{\text{GRP}}(G)$, for some group G. Our next theorem says that the future group operator can simulate the past group operator (and vice versa) in the presence of future and past operators.

Theorem 4.2.2. *Let $\alpha \in$ LTL$_{\text{GRP}}(G)$. Then α is equivalent to a formula $\hat{\alpha}$, where $\hat{\alpha}$ do not contain any past group operator.*

Proof. We replace all past group operators by future group operators as follows. Let $\beta := G^P_g\langle\phi_1, \ldots, \phi_k\rangle$ be a formula. Assume that all the formulas $\phi_1, ..., \phi_k$ do not contain any past group operator. We claim that β is equivalent to

$$\hat{\beta} = \bigvee_{l,j\in[k]} G^F_{g_l}\langle\phi_1, \ldots, \phi_k\rangle \wedge \check{\mathrm{P}}\,\check{\mathrm{H}}\, G^F_{g.g_j^{-1}.g_l}\langle\phi_1, \ldots, \phi_k\rangle \wedge \Gamma_j$$

We use Figure 5 to explain the equivalence. Let u be a word and $i \leq |u|$ such that $u, i \models \beta$. Then $\prod_{r=1}^{i} \Gamma(u,r) = g$ (the group value at a point is denoted by Γ. See Preliminaries chapter 2). There exists some $l, j \in [k]$ such that $\Gamma(u,i) = g_j$, $\prod_{r=i}^{|u|} \Gamma(u,r) = g_l$ and $\prod_{r=1}^{|u|} \Gamma(u,r) = g.g_j^{-1}.g_l$.

Therefore $u, i \models \beta$ iff $u, i \models \hat{\beta}$. □

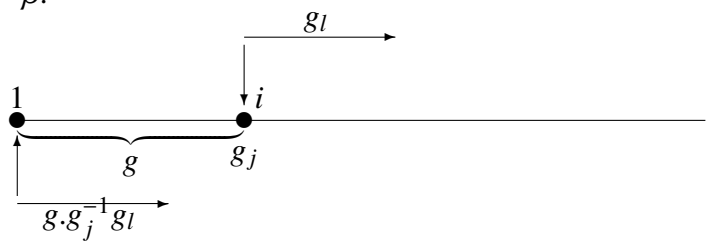

Figure 5: Future Group modality can simulate Past Group modality

Observe that the formula $\hat{\beta}$ in the proof above is an impure formula, even if β was one. This takes us to the next section, where we show that any formula in LTL$_{\text{GRP}}(G)$ can be written as a boolean combination of pure formulas.

4.2.1 *Separation property of* LTLGRP

We say that a formula can be *separated* if it can be written as a boolean combination of pure past, pure present and pure future formulas. We say that a logic satisfies the *separation property* [Gab87] if all formulas in that logic can be separated. Gabbay [Gab87] showed that LTL satisfies the separation property.

We next show that the logic LTL$_{\text{GRP}}(G)$, for a finite group G has the separation property. The proof is given by a series of technical lemmas. The translations given below in Lemma 4.2.3 is the base case for the full proof.

Lemma 4.2.3. *The following formulas can be separated.*

1. $G^F_g\langle(a\check{\mathrm{S}}b \wedge \beta) \vee \phi_1, \ldots, \phi_k\rangle$
2. $G^F_g\langle(G^P_{g'}\langle\gamma_1, ..., \gamma_k\rangle \wedge \beta) \vee \phi_1, \ldots, \phi_k\rangle$
3. $\alpha_1\check{\mathrm{U}}(G^P_{g'}\langle\gamma_1, ..., \gamma_k\rangle \wedge \alpha_2)$
4. $(G^P_{g'}\langle\gamma_1, ..., \gamma_k\rangle \vee \alpha_2)\check{\mathrm{U}}\alpha_1$
5. $G^P_g\langle(a\check{\mathrm{U}}b \wedge \beta) \vee \phi_1, \ldots, \phi_k\rangle$
6. $G^P_g\langle(G^F_{g'}\langle\gamma_1, ..., \gamma_k\rangle \wedge \beta) \vee \phi_1, \ldots, \phi_k\rangle$
7. $\alpha_1\check{\mathrm{S}}(G^F_{g'}\langle\gamma_1, ..., \gamma_k\rangle \wedge \alpha_2)$
8. $(G^F_{g'}\langle\gamma_1, ..., \gamma_k\rangle \vee \alpha_2)\check{\mathrm{S}}\alpha_1$

Proof. **(1):** Consider a word u as given in Figure 6. Let x be the position such that $u, x \models G^F_g\langle(a\check{\mathrm{S}}b \wedge \beta) \vee \phi_1, \ldots, \phi_k\rangle$. Consider positions y, z such that $x < y < z$ and $u, y \models b$ and $u, l \models a$ for all l, where $y < l < z$. Moreover let $u, z \models \beta$. That is z is where $a\check{\mathrm{S}}b \wedge \beta$ is true. Observe that (y, z) is a block of states which satisfy $a\check{\mathrm{S}}b$ formula. Our idea is to get the group value computed in this interval. We consider the case where x does not satisfy the formula $a\check{\mathrm{S}}b$ (the solution can be easily modified to the case when x satisfies $a\check{\mathrm{S}}b$).

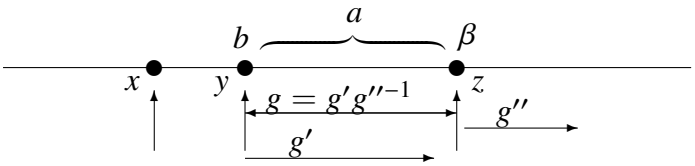

Figure 6: Timeline for $a\check{\mathrm{S}}b \wedge \beta$

Let $\Theta = \langle\phi_1, ..., \phi_k\rangle$. We now give a formula ψ_g which is true at all positions which satisfy $b \wedge a\check{\mathrm{U}}\beta$ and the group value computed for the block of as until β is g.

$$\psi_g := \bigvee_{g'=g.g''} b \wedge (a\check{\mathrm{U}}(\beta \wedge G^F_{g''}\Theta)) \wedge G^F_{g'}\Theta$$

Thus ψ_g is true at y iff the group value in the interval (y, z) is g. Let $\gamma = \neg(b \wedge a\check{\mathrm{U}}\beta)$. Now the following formula is equivalent to the original formula.

$$G^F_g\langle(\gamma \wedge \phi_1) \vee \psi_{g_1}, ..., (\gamma \wedge \phi_k) \vee \psi_{g_k}\rangle$$

The formula evaluates ϕ_is only when the position satisfy γ. Otherwise one of the ψ_{g_i}s would have calculated the group value for the entire block.

(2,3,4): Consider the word u given in Figure 7. Let us assume that the position x is such that

$$u, x \models G^P_{g_i}\langle\gamma_1, \ldots \gamma_k\rangle \ \wedge \ \Gamma_l \ \wedge \ G^F_{g_{j'}}\langle\gamma_1, \ldots, \gamma_k\rangle$$

Take $h = g_l.g_{j'}$ and let $u, y \models G^P_{g'}\langle\gamma_1, ..., \gamma_k\rangle$. Then the group value of the interval (x, y) is $m = g_i^{-1}g'$. Therefore the group value of points from y onwards is $m^{-1}h$ which is $g'^{-1}g_ih$ Then any point in the future will satisfy $G^P_{g'}\langle\gamma_1, ..., \gamma_k\rangle$ iff it also satisfies $G^F_{g'^{-1}g_ih}\langle\gamma_1, ...\gamma_k\rangle$. That is

$$u, y \models G^P_{g'}\langle\gamma_1, ..., \gamma_k\rangle \;\Leftrightarrow\; u, y \models G^F_{g'^{-1}g_ih}\langle\gamma_1, ...\gamma_k\rangle$$

This lets us replace the past group operator with a future group operator and vice versa.
(5,6,7,8): These formulas are got by replacing past operators with future operators and vice versa in the formulas in 1,2,3,4. By the same arguments above we can show that these formulas can also be separated.

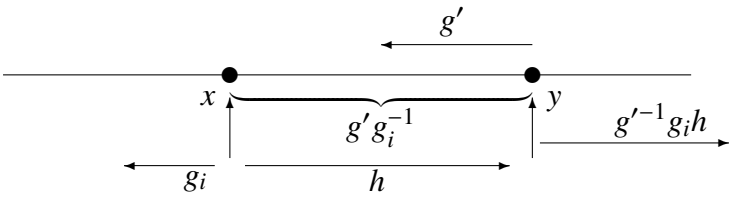

Figure 7: Timeline for $G^P_{g'}\langle...\rangle$

□

Lemma 4.2.4. *The following translations are equivalent.*

1. $\alpha_1 \check{\mathrm{S}}(\alpha_2 \vee \alpha_3) \equiv \alpha_1 \check{\mathrm{S}}\alpha_2 \vee \alpha_1 \check{\mathrm{S}}\alpha_3$
2. $(\alpha_1 \wedge \alpha_2)\check{\mathrm{S}}\alpha_3 \equiv \alpha_1 \check{\mathrm{S}}\alpha_3 \wedge \alpha_2 \check{\mathrm{S}}\alpha_3$
3. $\neg(a\check{\mathrm{S}}b) \equiv (\neg b\check{\mathrm{S}}\neg a) \vee \check{\mathrm{H}}\neg b$
4. $\neg G^P_g\langle\phi_1, \ldots, \phi_k\rangle \equiv \bigvee_{g \neq g_i} G^P_{g_i}\langle\phi_1, \ldots, \phi_k\rangle$
5. $\alpha_1 \check{\mathrm{U}}(\alpha_2 \vee \alpha_3) \equiv \alpha_1 \check{\mathrm{U}}\alpha_2 \vee \alpha_1 \check{\mathrm{U}}\alpha_3$
6. $(\alpha_1 \wedge \alpha_2)\check{\mathrm{U}}\alpha_3 \equiv \alpha_1 \check{\mathrm{U}}\alpha_3 \wedge \alpha_2 \check{\mathrm{U}}\alpha_3$
7. $\neg(a\check{\mathrm{U}}b) \equiv (\neg b\check{\mathrm{U}}\neg a) \vee \check{\mathrm{G}}\neg b$
8. $\neg G^F_g\langle\phi_1, \ldots, \phi_k\rangle \equiv \bigvee_{g \neq g_i} G^F_{g_i}\langle\phi_1, \ldots, \phi_k\rangle$

Proof. The correctness of (4) and (8) follows from the semantics of the group operator. The rest of the statements are standard LTL equivalence statements. These can be found in [Gab87]. □

Lemma 4.2.3 and Lemma 4.2.4 let us replace past operators nested inside future operators by future operators. Observe that there are dual Lemmas where the past operator is replaced by the future operator and vice versa. Using these two lemmas we give a series of lemmas to show that formulas in LTLGRP can be separated. These lemmas are proved using induction on the structure of the formula.

Lemma 4.2.5. *1. Let a, b be propositional formulas. Let α, β and $\phi_1, ..., \phi_k$ be formulas where $\breve{\mathrm{U}}$ appears only in the subformula $a\breve{\mathrm{U}}b$. Then $\alpha\breve{\mathrm{S}}\beta$, $G_g^P\langle\phi_1, \ldots, \phi_k\rangle$ can be separated.*

2. Let a, b be propositional formulas. Let α, β and $\phi_1, ..., \phi_k$ be formulas where $\breve{\mathrm{S}}$ appears only in the subformula $a\breve{\mathrm{S}}b$. Then $\alpha\breve{\mathrm{U}}\beta$, $G_g^F\langle\phi_1, \ldots, \phi_k\rangle$ can be separated.

Proof. Observe that the (ii)nd statement is the (i)st statement with past modalities replaced by future modalities and vice versa. We prove (i) now.
Gabbay [Gab87] shows how to separate the formula $\alpha\breve{\mathrm{S}}\beta$. So let us consider the formula $G_g^P\langle\phi_1, \ldots, \phi_k\rangle$. Let each ϕ_i be a boolean combination of $a\breve{\mathrm{U}}b$ and propositions. To rewrite $G_g^P\langle\phi_1, \ldots, \phi_k\rangle$ as boolean combination of pure Future and pure Past formulas, we apply the transformations given in Lemma 4.2.3 repeatedly which gives us a separated formula. □

Lemma 4.2.6. *1. Let $a_1, ..., a_k$ be propositional formulas. Let α, β and $\phi_1, ..., \phi_k$ be formulas having only the modality $G_g^F\langle a_1, ..., a_k\rangle$. Then $\alpha\breve{\mathrm{S}}\beta$, $G_{g'}^P\langle\phi_1, \ldots, \phi_k\rangle$ can be separated.*

2. Let $a_1, ..., a_k$ be propositional formulas. Let α, β and $\phi_1, ..., \phi_k$ be formulas having only the modality $G_g^P\langle a_1, ..., a_k\rangle$. Then $\alpha\breve{\mathrm{U}}\beta$, $G_{g'}^F\langle\phi_1, \ldots, \phi_k\rangle$ can be separated.

Proof. Repeated application of Lemma 4.2.3 and Lemma 4.2.4 give us a separated formula. □

We now look at formulas where an until modality (since modality) is nested inside a past modality (future modality).

Lemma 4.2.7. *1. Let $a_1, ..., a_n, b_1, ..., b_n$ be propositional formulas. Let α, β and $\phi_1, ..., \phi_k$ be formulas where $\breve{\mathrm{U}}$ appears only in subformulas of the form $U_i = a_i\breve{\mathrm{U}}b_i$, for all $i \leq n$. Then $\alpha\breve{\mathrm{S}}\beta$, $G_g^P\langle\phi_1, \ldots, \phi_k\rangle$ can be separated.*

2. Let $a_1, ..., a_n, b_1, ..., b_n$ be propositional formulas. Let α, β and $\phi_1, ..., \phi_k$ be formulas where $\breve{\mathrm{S}}$ appears only in subformulas of the form $S_i = a_i\breve{\mathrm{S}}b_i$, for all $i \leq n$. Then $\alpha\breve{\mathrm{U}}\beta$, $G_g^F\langle\phi_1, \ldots, \phi_k\rangle$ can be separated.

Proof. We prove (i) and claim that the proof for (ii) is similar. When $\psi = \alpha S \beta$, this can be separated by the arguments of Gabbay. So let $\psi = G_g^P\langle\phi_1, \ldots, \phi_k\rangle$. Let $\{U_1, ..., U_n\}$ be the n Until formulas used in the ϕ_is. We first replace the Until formulas $U_1, ..., U_{n-1}$ by new propositions $p_1, ..., p_{n-1}$. Let the new formula be called $\hat{\psi}$. By Lemma 4.2.5 we know that we can find a separated formula equivalent to $\hat{\psi}$. Now replace p_{n-1} in the formula by U_{n-1} and again apply Lemma 4.2.5. Observe that we did not introduce any new Untils when we separated. After n rounds we get a formula which is separated. □

Lemma 4.2.8. *1. Let $a_1, ..., a_n, b_1, ..., b_n$ be propositional formulas. Let α, β and $\phi_1, ..., \phi_k$ be formulas having only the modality $G_g^F\langle a_1, ..., a_k\rangle$. Then $\alpha\breve{\mathrm{S}}\beta$, $G_{g'}^P\langle\phi_1, \ldots, \phi_k\rangle$ can be separated.*

2. *Let $a_1, ..., a_n, b_1, ..., b_n$ be propositional formulas. Let α, β and $\phi_1, ..., \phi_k$ be formulas having only the modality $G_g^P\langle a_1, ..., a_k\rangle$. Then $\alpha \breve{\mathrm{U}} \beta$, $G_{g'}^F\langle \phi_1, \ldots, \phi_k\rangle$ can be separated.*

Proof. The proof is similar to the proof of Lemma 4.2.7. In (i) we replace $\forall i < n, G_g^F\langle a_1, ..., a_k\rangle$ by new propositions p_i. We then apply 4.2.5 and continue as in the proof of Lemma 4.2.7. □

Lemma 4.2.9. *1. Let $a_1, ..., a_n, b_1, ..., b_n$ be propositional formulas. Let α, β be formulas having only the modality $\forall i \leq n : U_i = a_i \breve{\mathrm{U}} b_i s$ or $G_g^F\langle a_1, ..., a_k\rangle$. Then $\alpha \breve{\mathrm{S}} \beta$, $G_{g'}^P\langle \phi_1, \ldots, \phi_k\rangle$ can be separated.*

2. *Let $a_1, ..., a_n, b_1, ..., b_n$ be propositional formulas. Let α, β be formulas having only the modality $\forall i \leq n : U_i = a_i \breve{\mathrm{S}} b_i s$ or $G_g^P\langle a_1, ..., a_k\rangle$. Then $\alpha \breve{\mathrm{U}} \beta$, $G_{g'}^F\langle \phi_1, \ldots, \phi_k\rangle$ can be separated.*

Proof. The proof is by combining the two Lemma 4.2.7 and Lemma 4.2.8. □

Now we look at formulas having future (past) modalities but without any past (future) modality nested inside a future or a past modality. That is no modality is nested inside a Future (Past) modality, but Past and Future modalities can be nested with Past (Future) modality.

Lemma 4.2.10. *1. Let $a_1, ..., a_k, b_1, ..., b_k$ be propositional formulas. Let α be a formula such that the Future modalities are of the form $\forall i \leq n : U_i = a_i \breve{\mathrm{U}} b_i s$ or $G_g^F\langle a_1, ..., a_k\rangle$. Then α can be separated.*

2. *Let $a_1, ..., a_k, b_1, ..., b_k$ be propositional formulas. Let α be a formula such that the Future modalities are of the form $\forall i \leq n : U_i = a_i \breve{\mathrm{S}} b_i s$ or $G_g^P\langle a_1, ..., a_k\rangle$. Then α can be separated.*

Proof. We prove (i) and claim that the proof for (ii) is similar. Let the Past depth be n. If $n = 0$ the claim is trivially true. When $n = 1$ the claim follows from Lemma 4.2.9. For depth $n > 1$ we repeatedly apply Lemma 4.2.9 to the most deeply nested Past modality. After each application of the Lemma the depth of the Past modality is reduced and hence after n steps we get a separated formula. □

Now we consider formulas which can have Future modalities nested inside the Past modality.

Lemma 4.2.11. *Let α be a formula such that no Past (Future) modality is nested inside a Future (Past) modality. Then α can be separated.*

Proof. The proof is by induction on the depth n of the Future (Past) modality. $n = 1$ was proved by Lemma 4.2.10. When $n > 1$, we replace all Future (Past) modalities at Future

(Past) depth ≥ 2 by new propositions p_i. Let the resultant formula be $\hat{\alpha}$. Observe that the Future (Past) depth of $\hat{\alpha}$ is one and hence can be separated by Lemma 4.2.10. Now replace all the p_is by the Future (Past) modalites we replaced them with. Observe that we have reduced the Future (Past) depth. We repeat the above process until we get a separated formula. □

Finally we show that any formula $\alpha \in$ LTLGRP can be separated.

Theorem 4.2.12. *Let α be an* LTLGRP(G) *formula. Then α can be separated.*

Proof. This involves induction on the alternation depth n of the formula α. $n = 1$ was proved in Lemma 4.2.11. When $n > 1$, we replace the modalities by propositions such that we get a formula $\hat{\alpha}$ which is of alternation depth one. Lemma 4.2.11 will give a separated $\hat{\alpha}'$ formula equivalent to $\hat{\alpha}$. Now replace the propositions in $\hat{\alpha}'$ with the modalities we had earlier replaced with. The formula we get is of alternation depth lesser than α. Hence we can repeat the procedure until we get a formula which is separated. □

Corollary 4.2.13. *Let α be an* LTLGRP *formula. Then α can be separated.*

Proof. From Theorem 4.2.1 we know that there exists an equivalent formula $\hat{\alpha} \in$ LTLGRP(G) for some finite group G. Then Theorem 4.2.12 gives us separation. □

Corollary 4.2.14. *Every* LTLGRP *formula is initially equivalent to a formula with only future modalities.*

Proof. Let α be an LTLGRP formula. By Theorem 4.2.12 α can be separated. We can now replace the past formulas with *false* since all statements regarding past are false at the 1^{st} position. The resultant formula which is equivalent to α now contains only future modalities. □

4.2.2 *Expressive Completeness of* LTLGRP

Lemma 4.2.15. LTLGRP *has separation property iff it is expressively complete for FO*GRP*[<].*

Proof. ($\Leftarrow$): Let α be an LTLGRP formula. We can now write a first order logic formula, $\alpha'(x)$ on free variable x, such that it is equivalent to α. First order logic formulas can be separated using relativization. This can be proved by induction on the structure of the formula. The atomic case is trivial. Formulas of form $\exists y \phi(x, y)$ can be replaced by

$$\exists y((y < x) \wedge \phi(x, y)) \vee \phi(x, x) \vee (\exists y(y > x \wedge \phi(x, y)))$$

Now a formula of type $\exists y(y > x \wedge \phi(x, y))$ can be replaced by a pure future LTLGRP formula (since LTLGRP is expressively complete for FOGRP[<]). Similarly we can replace

pure past and pure present FOGRP[<] formulas by pure past and pure present LTLGRP formulas. A similar proof can be given for group quantifiers too.
Now since LTLGRP is expressively complete for FOGRP[<] each of the separated formulas can be replaced with LTLGRP formulas. This gives us a separated formula.
($\Rightarrow$): We show that for an FOGRP[<] formula with one free variable we can give an equivalent LTLGRP formula. Let $P_1, ..., P_n$ be the unary predicates. The proof is by induction on the quantifier depth. For the base case we assume formulas with no quantifiers. This consists of boolean combination of formulas of the form $P_i(x)$. The translation of this formula will be boolean combination of formulas of the form p_i.
Now let us assume that all FOGRP[<] formulas with one free variable and of quantifier depth $< k$ over any constant number of unary predicates (alphabet) can be converted into an LTLGRP formula. Let Q be a quantifier. Consider the formula $\psi(x) = Qy\ \phi(x,y)$ such that ϕ is of quantifier depth $< k$. We first remove x from ψ as follows. All subformulae of the form $x = x, x < x, x > x$ are replaced by $\top, \bot, \bot$ respectively. Now we rewrite ϕ as follows (here $\vec{v} \in \{0,1\}^n$):

$$\hat{\phi}(x) = \bigvee_{\vec{v}=\{0,1\}^n} ((\bigwedge_{i=1}^{n} P_i(x) \Leftrightarrow v_i) \Rightarrow \phi^{\vec{v}}(x))$$

Here $\phi^{\vec{v}}(x)$ replaces all occurrences of subformulas of the form $P_i(x)$ with $\top, \bot$ depending on v_i. Now the subformulas in each of $\phi^{\vec{v}}$ containing x will be of the form $x < z, x > z, x = z$, where z is some other variable in ϕ. We remove these formulae by introducing three new unary predicates $R_<, R_>, R_=$ and replacing $x\ op\ z$ by $R_{op}(z)$ (op $:= \{<, >, =\}$). The resultant formula $\phi^{\vec{v}}$ will not contain any occurrence of x. Moreover if we assume the interpretations for R_{op} it will be equivalent to the old formula. That is $w, i \models \psi(x)$ if and only if $w \models Qy\ \hat{\phi}(x)$ provided we interpret $R_<(z) = \top \Leftrightarrow i < z$, $R_>(z) = \top \Leftrightarrow i > z$ and $R_=(z) = \top \Leftrightarrow i = z$.
Case 1, $Q = \exists$: Since $\phi^{\vec{v}}(y)$ is a formula with one free variable and quantifier depth $< k$, we can apply the inductive hypothesis to get an LTLGRP formula γ with new propositions $r_<, r_>, r_=$. We now write $\beta = \mathrm{P}\gamma \vee \gamma \vee \mathrm{F}\gamma$. From Corollary 4.2.14 it follows that β can be separated into a boolean combination of pure past, pure present and pure future formulas. Finally in all the pure past formulas we replace $r_<, r_>, r_=$ with $\top, \bot, \bot$ respectively. Similarly one can replace all the r_{op} formulae with $\top, \bot$ in the pure future, and pure present formulae.
Case 2, $Q = Q^g_G$: Let $\psi^{\vec{v}} = Q^g_G y\ \langle \phi^{\vec{v}}_1(y), \ldots, \phi^{\vec{v}}_k(y) \rangle$. Since $\phi^{\vec{v}}_i(y)$s are formulas with one free variable and quantifier depth $< k$, we can apply the inductive hypothesis to get LTLGRP formulas ϕ_is with new propositions $r_<, r_>, r_=$.
Let us denote by $\Phi = \langle \phi_1, ..., \phi_k \rangle$. Then we can write $\beta = \bigvee_{i,j,l} G^P_{g_i}\Phi \wedge \Gamma_l \wedge G^F_{g_j}\Phi$, such that $g_i.g_l.g_j = g$.
From Corollary 4.2.14 β can be separated into a boolean combination of pure past, pure present and pure future formulas. Finally in all the pure past formulas we replace $r_<, r_>, r_=$ with $\top, \bot, \bot$ respectively. Similarly one can replace all the r_{op} formulae with $\top, \bot$ in the pure future, and pure present formulae.
So we have shown that the formula $\psi(x)$ has an equivalent LTLGRP formula. □

Lemma 4.2.15 along with Theorem 4.2.12 gives us that

Theorem 4.2.16. LTL$_{GRP}$ *is expressively complete for FO*$_{GRP}$*[<]*

As a corollary we get

Corollary 4.2.17. *Every FO*$_{GRP}$*[<] (FO*$_{MOD}$*[<]) formula with one free variable has an equivalent formula using three variables.*

Proof. Let $\phi(x)$ be an FO$_{GRP}$[<] (FO$_{MOD}$[<]) formula. By Theorem 4.2.16 we know that there exists an equivalent LTL$_{GRP}$ (LTL$_{MOD}$) formula ψ. We now inductively built an FO$_{GRP}$[<] (FO$_{MOD}$[<]) formula from ψ as follows. The translation, $t : \text{LTL}_{GRP}\times\{x, y, z\} \to$ FO$_{GRP}$[<] is inductively given. Let $t_x(\alpha)$, $t_y(\alpha)$, $t_z(\alpha)$ denotes $t(\alpha, x)$, $t(\alpha, y)$, $t(\alpha, z)$ respectively. For a formula α we give the translation t_x by the Table 2. Clearly this

LTL$_{GRP}$	FO$_{GRP}$[<]
p	$P(x)$
$\alpha \vee \beta$	$t_x(\alpha) \vee t_x(\beta)$
$\neg\alpha$	$\neg t_x(\alpha(x))$
$\alpha \check{\mathrm{U}} \beta$	$\exists y\ (y > x) \wedge t_y(\beta(y)) \wedge \forall z\ (x < z < y) \Rightarrow t_z(\alpha(z))$
$\alpha \check{\mathrm{S}} \beta$	$\exists y\ (y < x) \wedge t_y(\beta(y)) \wedge \forall z\ (y < z < x) \Rightarrow t_z(\alpha(z))$
$G_g^F\langle\phi_1, ..., \phi_k\rangle$	$Q_G^g y\ \langle(x < y) \wedge t_y(\phi_1(y)), \ldots, (x < y) \wedge t_y(\phi_k(y))\rangle$
$G_g^P\langle\phi_1, ..., \phi_k\rangle$	$Q_G^g y\ \langle(x > y) \wedge t_y(\phi_1(y)), \ldots, (x > y) \wedge t_y(\phi_k(y))\rangle$

Table 2: Translation from LTL$_{GRP}$ formula to FO$_{GRP}$[<] formula.

translation uses only three variables and hence we get an equivalent FO$_{GRP}$[<] (FO$_{MOD}$[<]) formula in three variables. □

Note that we can give a similar proof for the following claims.

Theorem 4.2.18. *Let G be a finite group. Then the following holds.*

- LTL$_{GRP}$*(G) has separation property iff it is expressively complete for FO*$_{GRP}$*(G)[<].*
- LTL$_{GRP}$*(G) is expressively complete for FO*$_{GRP}$*(G)[<].*
- *Every FO*$_{GRP}$*(G) formula with one free variable has an equivalent formula using three variables.*

4.3 UTL AND TWO VARIABLE FRAGMENT OF FO[<]

Here we show that TL[F,P,GRP] is expressively complete for the two variable logic fragment of FO$_{GRP}$[<], (written as FO^2$_{GRP}$[<]).

Theorem 4.3.1. TL[F,P,GRP] *is expressively complete for* FO^2GRP[<].

Proof. The translation is recursive. For the atomic formulas, boolean combinations and existential formulas we would refer the readers to follow the proof by Etessami et al [EVW02]. We give here the translation for the group quantifier (A similar translation can be given for the modulo quantifiers). Let

$$\phi(x) := Q^g_G y \, \langle \phi^1(x,y), \ldots, \phi^k(x,y) \rangle$$

Each of the $\phi^j(x,y)$, for $j \leq k$ can be rewritten as

$$\phi^j(x,y) := \tau^j(\gamma^j_1(x,y), ..., \gamma^j_r(x,y), \alpha^j_1(x), ..., \alpha^j_s(x), \beta^j_1(y), ..., \beta^j_t(y))$$

Here τ^js are boolean propositional formula. γ^j_is are order formulas of the form $x < y, x = y, x > y$. α^j_is and β^j_is are formulas whose quantifier depth is less than the quantifier depth of $\phi(x)$. Moreover α^j_is and β^j_is are of type atomic or existential or group quantified. We first take out the α^j_is outside the group quantifier. For a vector $\vec{v} = (v_1, \ldots, v_{sk}) \in \{0,1\}^{sk}$ we define

$$\psi^j_{\vec{v}}(x,y) = \tau^j(\gamma^j_1(x,y), ..., \gamma^j_r(x,y), v_{j,1}, ..., v_{j,s}, \beta^j_1(y), ..., \beta^j_t(y))$$

Let ψ_g for a $g \in G$ be $Q^g_G y \, \langle \psi^1_{\vec{v}}(x,y), \ldots, \psi^k_{\vec{v}}(x,y) \rangle$. Then we can rewrite ϕ as:

$$\phi(x) := \bigvee_{v \in \{0,1\}^{ks}} \Big(\bigwedge_{j \leq k, i < s} \alpha^j_i(x) \Leftrightarrow v_{j,i} \Big) \wedge \psi^g(x)$$

Observe that the γ^j_is are order formulas. Let $\Gamma = \{x < y, x = y, x > y\}$, be the set of all order relations between x and y. For any order relation, $o \in \Gamma$, γ^j_is will be evaluated to $\{T, F\}$. Let ψ^o_g be the formula got by replacing γ^j_i with T/F in $\psi_g(x)$ depending on the order o. Observe that x does not appear free in ψ^o_g. Thus we get

$$\phi(x) := \bigvee_{v \in \{0,1\}^{ks}} \Big(\bigwedge_{j \leq k, i < s} \alpha^j_i(x) \Leftrightarrow v_{j,i} \Big) \wedge \bigvee_{g_1 g_2 g_3 = g} \psi^{y<x}_{g_1} \wedge \psi^{x=y}_{g_2} \wedge \psi^{y>x}_{g_3}$$

Since formulas ψ^o_g do not contain free variables, and since β_is have quantifier depth less than the quantifier depth of $\phi(x)$, equivalent TL[F,P,GRP] formulas exists. Formulas $\alpha^j_i(x)$ have equivalent TL[F,P,GRP] formulas, since their quantifier depths are less than the quantifier depth of $\phi(x)$. Hence we get an TL[F,P,GRP] formula equivalent to ϕ.

□

By following the above arguments one can also show.

Theorem 4.3.2. TL[F,P,X,Y,GRP] *is expressively complete for* FO^2GRP[<, *succ*].

4.4 DISCUSSION

In this chapter we saw that the counting versions of LTL are expressively equivalent to the counting versions of FO[<], namely LTL$_{MOD}$ is expressively as powerful as FO$_{MOD}$[<]. Infact these results can be extended to show that LTL$_{GRP}$ can express all languages expressible in FO$_{GRP}$[<]. We could also show that LTL$_{MOD}$ (or even LTL$_{GRP}$) satisfies the *separation* property, that is any formula in LTL$_{MOD}$ (LTL$_{GRP}$) is equivalent to a formula which is a boolean combination of pure future, pure past and present formulas. This separation property also shows that any LTL$_{GRP}$ formula is initially equivalent to a pure future formula.

The significance of the above equivalence result is as follows. We know that FO$_{GRP}$[<] can express any regular language. Thus, now we can write temporal logic formulas which can express any regular language property. This coupled with our PSPACE algorithm in the next chapter can be of use in verification purposes.

Baziramwabo, McKenzie and Thérien [BMT99] had given an algebraic characterization for LTL$_{GRP}$. It then follows, from the algebraic characterization of FO$_{GRP}$[<], that LTL$_{GRP}$ and FO$_{GRP}$[<] define the same set of languages. The following Figure 8 puts our contribution in perspective.

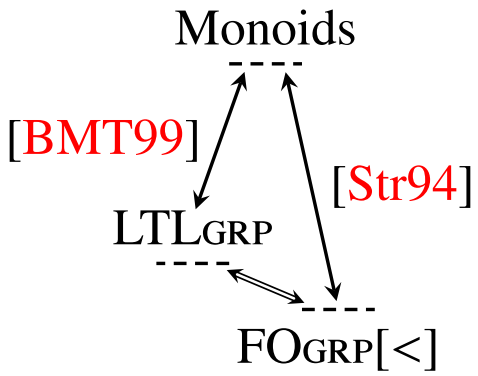

Figure 8: Equivalence of LTL and FO[<] in the presence of group quantifiers. The arrows denote language equivalence. The double arrow is our contribution

We also saw that the two variable fragment of first order logic, FO^2$_{GRP}$[<, *succ*] also has an equivalent temporal logic fragment, UTL extended with group operators, which is TL[F,P,X,Y,GRP]. The language equivalence of these various fragments of LTL$_{GRP}$ and FO$_{GRP}$[<] are captured in Table 3.

Temporal logic	First order logic
$\mathrm{LTL}_{\mathrm{GRP}}$	$\mathrm{FO}_{\mathrm{GRP}}[<]$
$\mathrm{LTL}_{\mathrm{GRP}}(G)$	$\mathrm{FO}_{\mathrm{GRP}}(G)[<]$
$\mathrm{TL}[\mathrm{F}, \mathrm{P}, \mathrm{GRP}]$	$\mathrm{FO}^2{}_{\mathrm{GRP}}[<]$
$\mathrm{TL}[\mathrm{F}, \mathrm{P}, \mathrm{X}, \mathrm{Y}, \mathrm{GRP}]$	$\mathrm{FO}^2{}_{\mathrm{GRP}}[<, succ]$
$\mathrm{TL}[\mathrm{F}, \mathrm{P}, \mathrm{GRP}(G), \mathrm{LEN}]$	$\mathrm{FO}^2{}_{\mathrm{GRP}}(G)[<, \equiv]$
$\mathrm{TL}[\mathrm{F}, \mathrm{P}, \mathrm{X}, \mathrm{Y}, \mathrm{GRP}(G), \mathrm{LEN}]$	$\mathrm{FO}^2{}_{\mathrm{GRP}}(G)[<, succ, \equiv]$
$\mathrm{TL}[\mathrm{F}, \mathrm{P}, \mathrm{MOD}]$	$\mathrm{FO}^2{}_{\mathrm{MOD}}[<]$
$\mathrm{TL}[\mathrm{F}, \mathrm{P}, \mathrm{X}, \mathrm{Y}, \mathrm{MOD}]$	$\mathrm{FO}^2{}_{\mathrm{MOD}}[<, succ]$
$\mathrm{TL}[\mathrm{F}, \mathrm{P}, \mathrm{MOD}(q), \mathrm{LEN}]$	$\mathrm{FO}^2{}_{\mathrm{MOD}}(q)[<, \equiv]$
$\mathrm{TL}[\mathrm{F}, \mathrm{P}, \mathrm{X}, \mathrm{Y}, \mathrm{MOD}(q), \mathrm{LEN}]$	$\mathrm{FO}^2{}_{\mathrm{MOD}}(q)[<, succ, \equiv]$
$\mathrm{TL}[\mathrm{F}, \mathrm{P}, \mathrm{LEN}]$	$\mathrm{FO}^2[<, \equiv]$
$\mathrm{TL}[\mathrm{F}, \mathrm{P}, \mathrm{X}, \mathrm{Y}, \mathrm{LEN}]$	$\mathrm{FO}^2[<, succ, \equiv]$

Table 3: Expressive power of various fragments of $\mathrm{LTL}_{\mathrm{GRP}}$

5

LTLGRP SATISFIABILITY

5.1 INTRODUCTION

In this chapter we show that satisfiability of LTLGRP is in PSPACE even when using the *succinct* notation (ie. binary notation to represent modulo counting and symmetric groups for the group operation). Unlike Serre [Ser04], we do not use alternating automata but ordinary NFA and the standard "formula automaton" construction in our decision procedure. Next we give a corresponding lower bound. Our second result shows that satisfiability of TL[F,MOD(2)] is PSPACE-hard. Since TL[F] is NP-complete, this shows that modulo counting is powerful.

We then look at a weaker logic and show that the satisfiability problem of the logic TL[F,P,LEN], is in Σ_3^P, the third level of the polynomial-time hierarchy, again irrespective of whether we use unary or binary notation. We also give a corresponding lower bound for the logic TL[F,LEN].

Finally we look at satisfiability of the logic TL[DUR], the logic where there are no temporal logic modalities other than one which can count propositions modulo a number. We show that satisfiability of this logic is NP-complete.

Table 4 gives a summary of the results of this chapter. Note that in the table the lower bound results carry over from top to bottom and from left to right, whereas the upper bound results carry over from right to left and from bottom to top.

TL	[]	[LEN]	[DUR]	[MOD]	[GRP]
[]	NP-hard [Coo71]	NP-C	**NP**	?	?
[F]	NP[ON80]	$\mathbf{\Sigma_3^P}$ **-complete**	**PSPACE-hard**	PSPACE-C	PSPACE-C
[F, X]	PSPACE-hard [SC85]	PSPACE-C	PSPACE-C	PSPACE-C	PSPACE-C
[F, X, U]	PSPACE[Pnu77c]	PSPACE-C	PSPACE-C	PSPACE-C	**PSPACE**

Table 4: Upper and lower bound results for satisfiability of various temporal logics. Except for those marked with ? the satisfiability for the rest of the logics are fully identified. The entries which are in **bold** are the results of this chapter.

5.2 MODULO COUNTING

5.2.1 *A* PSPACE *upper bound*

Our first main theorem shows that the upper bound for LTL satisfiability can be extended to include the modulo and group counting computations, even when specified in succinct notation.

Theorem 5.2.1. *If an* LTLGRP^{bin}*formula* α_0 *is satisfiable then there exists a satisfying model of size exponential in* α_0

Proof. The Fischer-Ladner closure of a formula α_0 [FL79], denoted by $CL(\alpha_0)$, is the least Δ such that

1. $\alpha_0 \in \Delta$.
2. If $\alpha_1 \vee \alpha_2 \in \Delta$ then $\alpha_1, \alpha_2 \in \Delta$.
3. $\neg\alpha_1 \in \Delta$ iff $\alpha_1 \in \Delta$
4. If $\mathrm{X}\alpha \in \Delta$ then $\alpha \in \Delta$
5. If $\alpha_1 \mathrm{U} \alpha_2 \in \Delta$ then $\alpha_1, \alpha_2 \in \Delta$
6. The closure of $\mathrm{MOD}^P_{r,q}\alpha$ includes α and also has $\mathrm{MOD}^P_{s,q}\alpha$ for every s from 0 to $q-1$. (Notice that only one of these can be true in a state.)
7. The closure of $\mathrm{MOD}^F_{r,q}\alpha$ includes α and also has $\mathrm{MOD}^F_{s,q}\alpha$ for every s from 0 to $q-1$.
8. The closure of $G^F_h\langle\alpha_1, \ldots, \alpha_k\rangle$ includes $\alpha_1, \ldots, \alpha_k$ and also contains the formulae $G^F_{h'}\langle\alpha_1, \ldots, \alpha_k\rangle$ for every element $h' \in G$. (Only one of these can be true at a state.)
9. The closure of $G^P_h\langle\alpha_1, \ldots, \alpha_k\rangle$ includes $\alpha_1, \ldots, \alpha_k$ and also contains the formulae $G^P_{h'}\langle\alpha_1, \ldots, \alpha_k\rangle$ for every element $h' \in G$.

Observe that if $\alpha_0 \in \mathrm{LTLGRP}^{bin}$, $CL(\alpha_0)$ can be exponential in the size of α_0, unlike the usual linear size for LTL, since the constants r, q and h are written in succinct notation. A state of the tableau or formula automaton which we will construct is a maximal consistent subset of formulae from the closure of α_0. Let $\mathrm{MOD}^P_{r,q}\alpha$ be a formula in the Fischer-Ladner closure. However, only one of the potentially exponentially many formulae of the form $\mathrm{MOD}^P_{s,q}\alpha, 0 \leq s < q$ can consistently hold in a state. Similarly only one of the formulae of the form $\mathrm{MOD}^F_{s,q}\alpha, 0 \leq s < q$ or of the form $G^F_h\langle\alpha_1, \ldots, \alpha_k\rangle$, can consistently hold. So a state is also exponential in the size of α_0. Here is a formal argument, using induction on structure of α, that the set of states of the automaton M_α is $2^{O(|\alpha|)}$. We denote by $|q|$ and $|G|$ for the input size (binary notation) and by S_α the number of states in M_α.

1. $\alpha = p \in P$. This is trivial.
2. $\alpha = \beta \vee \gamma$. $S_\alpha = S_\beta \times S_\gamma \leq 2^{O(|\beta|+|\gamma|)}$ (By induction hypothesis)
3. $\alpha = \neg\beta$. This is just change of final states in M_β.
4. $\alpha = \beta \mathrm{U} \gamma$. $S_\alpha = S_\beta \times S_\gamma \leq 2^{O(|\beta|+|\gamma|)}$
5. $\alpha = \text{MOD}^P_{r,q}\beta$ Since any atom can have only one formula of this kind, $S_\alpha = S_\beta \times q \leq 2^{O(|\beta|+|q|)}$
6. $\alpha = \text{MOD}^F_{r,q}\beta$. This is similar to the case above.
7. $\alpha = G^P_h\langle\alpha_1, \ldots, \alpha_k\rangle$. Again any atom can have only one formula of this kind, $S_\alpha = S_{\alpha_1} \times \cdots \times S_{\alpha_k} \times card(G) \leq 2^{O(|\alpha_1|+\cdots+|\alpha_k|+|G|)}$.
8. $\alpha = G^F_h\langle\alpha_1, \ldots, \alpha_k\rangle$. This is again similar to the case above.

□

Corollary 5.2.2. LTLGRP^{bin} *satisfiability is in* PSPACE.

Proof. Since the formula automaton has exponentially many states, each state as well as the transition relation can be represented in polynomial space. Now we can guess and verify an accepting path in PSPACE [Eme90]. □

We now look at the model checking problem for the above logics.

Theorem 5.2.3. *The model checking problem for* LTLGRP^{bin} *is* PSPACE*complete.*

Proof. Let α_0 be a formula in LTLGRP^{bin}and M a Kripke structure. Theorem 5.2.1 shows that for a formula $\neg\alpha_0$ there is an exponential size model $M_{\neg\alpha_0}$ which captures the language defined by $\neg\alpha_0$. Verifying $M \models \alpha_0$ is equivalent to checking whether the intersection of the above automatas is non-empty. This can be done by a non-deterministic machine which uses space logarithmic in the size of both the models. The lower bound follows from the fact that model checking for LTL is PSPACE-hard. □

5.2.2 *Corresponding lower bound*

Next we consider the logic TL[F,MOD]. It can express properties which can be expressed by LTL but not by TL[F], for example $\mathrm{G}(p \Leftrightarrow \ell^P_{1,2})$ expresses alternating occurrences of p and $\neg p$. Our next result shows that the satisfiability problem for TL[F,MOD], even with unary notation, is PSPACE-hard.

Theorem 5.2.4. *The satisfiability problem for* TL*[*F,MOD(2)*] is* PSPACE*-hard, even with the modulo formulae restricted to counting propositions,*

Proof. Since the satisfiability problem for TL[X,F] is PSPACE-hard [SC85], giving a satisfiability preserving polynomial translation from formulas in TL[X,F] to TL[F,MOD(2)] will give us the required result. It is sufficient to give a polynomial-sized translation of the modality Xα. For each such formula, we introduce two new propositions p^E_α and p^O_α, and enforce the constraints below. Let *EvenPos* abbreviate $\ell^P_{0,2}$ and *OddPos* abbreviate $\ell^P_{1,2}$.

1. Consider an even position (a position where *EvenPos* is true). At that position, we ensure α is true if and only if p^E_α is true. Similarly at an odd position, α is true if and only if p^O_α is true. The following formula can ensure this.
$$\mathrm{G}\Big(\alpha \Leftrightarrow \big((\mathit{EvenPos} \Rightarrow p^E_\alpha) \wedge (\mathit{OddPos} \Rightarrow p^O_\alpha)\big)\Big)$$
2. Let m be some even position. Then what we want is that the number of occurrences of p^E_α from the beginning of the word to m should be even. Similarly for an odd position, we want the number of occurrences of p^O_α from the beginning to it is odd. The following formula takes care of this condition.
$$\mathrm{G}\Big((\mathit{EvenPos} \Rightarrow \mathrm{MOD}^P_{0,2} p^E_\alpha) \wedge (\mathit{OddPos} \Rightarrow \mathrm{MOD}^P_{0,2} p^O_\alpha)\Big)$$
3. Then all occurrences of the formula Xα are replaced by the following formula.
$$(\mathit{EvenPos} \Rightarrow p^O_\alpha) \wedge (\mathit{OddPos} \Rightarrow p^E_\alpha).$$

Let us analyze what these conditions enforce. We claim the following.

Claim 5.2.5. *For an odd z, we have*
$$w, z \models \mathrm{X}\alpha \Leftrightarrow w, z \models p^E_\alpha$$
Similarly, for an even z, we have
$$w, z \models \mathrm{X}\alpha \Leftrightarrow w, z \models p^O_\alpha$$

Proof. Consider a word w which satisfies the above two conditions. For a point $z \leq |w|$ and a proposition p, let $C(z, p)$ be the number of occurrences of p at points $\leq z$. That is
$$C(z, p) = |\{i \leq z \mid w, i \models p\}|$$
Let us first look at the proposition p^E_α. From condition (2), we know that the count $C(z, p^E_\alpha)$ is even at all even points z. Let $z > 2$ be an even point where α is true. By (2), the counts $C(z, p^E_\alpha)$ and $C(z-2, p^E_\alpha)$ should be even. But from (1) we have that $w, z \models p^E_\alpha$, and hence $C(z, p^E_\alpha) > C(z-2, p^E_\alpha)$. Since both these counts are even, the word w is forced to satisfy the condition, $w, z-1 \models p^E_\alpha$. Let us now look at the other direction. Let us assume that for an even z, we have $w, z-1 \models p^E_\alpha$. From (2) we have that the counts $C(z, p^E_\alpha)$ and $C(z-2, p^E_\alpha)$ should be even and from our assumption $C(z, p^E_\alpha) > C(z-2, p^E_\alpha)$ and hence $w, z \models p^E_\alpha$. From (1) it then follows that $w, z \models \alpha$.

So at an *odd* position, Xα holds precisely when p^E_α holds. Symmetrically, at an *even* position, Xα holds if and only if p^O_α holds. □

From the above claim it follows that condition (3) is necessary and sufficient. Note that $X\alpha$ is replaced by a formula of size $|\alpha| + c$ for a constant c. With one such translation for every X modality, the reduction is linear. □

5.3 LENGTH MODULO COUNTING

We now consider the weaker logic TL[F,LEN] (Refer to Preliminaries chapter 2 for the definition). So we can only count lengths rather than propositions, which was something we needed in the PSPACE-hardness proof in the previous section.

Note that the language $(ab)^*$ is in TL[F,LEN]. It is known [ON80, SC85, Sch02] that a satisfiable formula in TL[F] has a polynomial sized model. Unfortunately

Proposition 5.3.1. *TL[F,LEN] does not satisfy a polynomial model property.*

Proof. Let p_i be distinct primes (in unary notation) in the following formula:

$$\mathrm{F}((\ell^P_{0,p_1}) \wedge (\ell^P_{0,p_2}) \wedge \cdots \wedge (\ell^P_{0,p_n})).$$

Any model which satisfies this formula will be of length at least the product of the primes, which is $\geq 2^n$. □

We show that the satisfiability problem of TL[F,LEN] is in Σ^P_3, the third level of the polynomial-time hierarchy.

Let us first look at the sublogic TL[LEN]. We give a couple of technical lemmas concerning the logic TL[LEN] which will be crucial to our arguments later. Observe that the truth of a TL[LEN] formula in a word w at position i, does not depend on the letters at positions $j \neq i$. That is

Lemma 5.3.2. *Let $\alpha \in$ TL[LEN] and $w, i \models \alpha$. Then for all w' such that $w'[i] = w[i]$ we have $w', i \models \alpha$.*

Proof. We prove by induction on the structure of α. Clearly the claim holds for propositions and $\ell^P_{r,q}, \ell^F_{r,q}$ since the states have the same set of propositions. It is also easy to note that the claim is closed under conjunctions and negations. □

We now look at a non-trivial property of TL[LEN]. This property will be used later for our upper bound result.

Lemma 5.3.3. *Let α be a TL[LEN] formula. Then the following are equivalent.*

1. $\forall w \ ((|w| = n) \Rightarrow \exists k \leq n, \ (w,k) \models \alpha)$
2. $(\exists k \leq n, \forall w \ ((|w| = n) \Rightarrow \ (w,k) \models \alpha))$

Proof. $(2 \Rightarrow 1)$: This is trivial.
$(1 \Rightarrow 2)$: Assume (1) is true but (2) is false. Let $S = \{w \mid |w| = n\}$. Pick a $w_1 \in S$. By the hypothesis $\exists i \leq n,\ (w_1, i) \models \alpha$ and since the claim is false, there exists some $w_2 \in S$ such that $(w_2, i) \not\models \alpha$. If this is not true then we have a witness i, such that $\forall w \in S\ (w, i) \models \alpha$. Let u_i be the state at the i^{th} location of w_2. Replace the i^{th} state in w_1 by u_i without changing any other state in w_1. Call this new word w_3. Now by Lemma 5.3.2, $(w_3, i) \not\models \alpha$. Again by the hypothesis, $\exists j \leq n,\ (w_3, j) \models \alpha$. By the same argument given above, $\exists w_4$ such that $(w_4, j) \not\models \alpha$. We can replace the j^{th} state of w_3 by the the j^{th} state from w_4 which makes the resultant word not satisfy α at the j^{th} location. We continue doing the above procedure for n steps. Since n is finite after some finite occurrence of the above procedure, we will get a word v such that $\forall k \leq n,\ (v, k) \not\models \alpha$. But this implies the hypothesis is wrong and hence a contradiction. □

The above Lemma shows that an algorithm to check the first condition, need only check the second condition. The second condition can be easily verified by a Σ_2^P machine. We are now going to use the above Lemma to solve the following problem.

Definition 5.3.4. *(BlockSAT): Given a* TL[LEN] *formula α and two numbers m, n in binary, the problem BlockSAT is to check whether there exists a model w of size $m + n$ such that $(w, m) \models G\alpha$.*

Lemma 5.3.5. *The problem BlockSAT is in Π_2^P.*

Proof. The algorithm takes as input a TL[LEN] formula α, along with two numbers m, n in binary. Observe that since n is in binary we cannot guess the entire model. The algorithm needs to check whether there exists a model w such that $|w| = m + n$ and for all k, where $m \leq k \leq m + n$, we have $w, k \models \alpha$. Take the complement of this statement, which is for all words w, where $|w| = m + n$, there exists a k, such that $m \leq k \leq m + n$ and $w, k \models \neg\alpha$. That is $\forall w, |w| = m + n \Rightarrow \exists k\ m \leq k \leq m + n,\ (w, k) \models \neg\alpha$. By the previous Lemma 5.3.3 we can check this condition by a Σ_2^P machine. Hence *BlockSAT* can be verified by a Π_2^P machine. □

5.3.1 *Length modulo counting - Upper bound*

We show that satisfiability of TL[F,LEN] can be checked in Σ_3^P, even in binary notation, showing that this restriction does buy us something.

Note that Σ_3^P is a subset of PSPACE, but it is not known whether this is a strict subset. Hence for all practical purposes we require polynomial space in checking for satisfiability and model checking.

Before proceeding into an algorithm, we need to introduce a few definitions. Let α be a formula over a set of propositions $\mathcal{P}$, $SubF(\alpha)$ its set of future subformulae. That is

$$SubF(\alpha) = \{F\beta \mid F\beta \text{ is a subformula of } \alpha\}$$

$prd(\alpha)$ the product over all elements of the set $\{n \mid (\delta \equiv r \mod n)$ is a subformula of $\alpha\}$. Let w be a word model. We define *witness index* in w for α as (assume $max\,\emptyset = 1$)

$$WI(w,\alpha) = \{max\{j \mid w, j \models \mathrm{F}\beta\} \mid \mathrm{F}\beta \in SubF(\alpha)\} \cup \{1\}$$

This is the set of all points j, which satisfy Fβ for a subformula Fβ of α and such that there is no point in the strict future of j which satisfy Fβ. A state at a witness index is called a *witness state*. For a subformula Fβ of α, we say β is witnessed at i if i is the future most point which satisfy β. In other words, $i = max\{j \mid w, j \models \mathrm{F}\beta\}$. Call all states other than witness states of w as *pad states* of w for α.

We define a model w to be *nice* for α if between any two consecutive witness states of w (for α) there are at most $prd(\alpha)$ number of pad states. We claim that if α is satisfiable then it is satisfiable in a nice model.

Theorem 5.3.6. *Let α be a satisfiable formula in* TL[F, LEN]. *Then there exists a nice model w which satisfy α.*

Proof. Since α is satisfiable, there exists a word w such that $w, 1 \models \alpha$. Let $prd(\alpha) = q$ and let $i, j \in WI(w,\alpha)$ such that $j - i - 1 > q$ and $\forall i < k < j\ k \notin WI(w,\alpha)$. Let us also assume that β_1, β_2 are witnessed at i and j respectively. Let w' be the first r positions in the subword $w[i+1, j]$ where $j - i - 1 \equiv r(\mod q)$ and $r < q$. We now construct a new model $\hat{w} = w[1,i]w'w[j,|w|]$. Consider the mapping

$$\nu(k) = \begin{cases} k & \text{if } k \leq i + r \\ k + j - i - r - 1 & \text{otherwise} \end{cases}$$

We now show for a subformula β of α that $\hat{w}, k \models \beta \Leftrightarrow w, \nu(k) \models \beta$ for all $k \leq |\hat{w}|$. The truth at a state depends only on the propositions at that state, the truth of future formulas and the truth of the length formulas. The claim now follows, since $\hat{w}[k] = w[\nu(k)]$ and $k \equiv \nu(k)(\mod q)$ and the witness states are preserved. That is $WI(\hat{w},\alpha) = \{\nu(k) \mid k \in WI(w,\alpha)\}$. □

Thus, a normal model of α will be of size less than or equal to $|SubF(\alpha)| \times prd(\alpha)$, which is of size exponential in α. So guessing the normal model is too expensive, but we can guess the witness states (the indices and propositions true at these states), which are polynomial, verify whether the F requirements are satisfied there, and verify if there are enough pad states to fill the gap between the witness states. We will argue that we can use a Π_2 oracle to verify the latter part. The proof is given below.

Theorem 5.3.7. *Satisfiability of* TL[F, LEN] *is in Σ_3^P, even if the syntax uses binary notation.*

Proof. Let α be satisfiable. We guess the following and use it to verify whether there exists a normal word satisfying these guesses.

1. Guess k positions, $1 = l_0 < l_1 < l_2 < ... < l_k$, where $k \leq |SubF(\alpha)|$ and $\forall i,\ (l_{i+1} - l_i - 1) \leq prd(\alpha)$. We have to verify that these are the indices where the future formulas are witnessed.

2. Guess $k+1$ sets of propositions $a_1, \ldots, a_k$. These form the witness states.

3. For each index $j \in [k+1]$, guess the set B_j of all β such that β is witnessed at l_j. Let $B = \{\beta \mid \mathrm{F}\beta \in SubF(\alpha)\}$. Then

$$B_j = \{\beta \mid \beta \in B \text{ and } \beta \text{ is witnessed at } l_j\}$$

Note that $\cup_{j \leq k} B_j$ can be a proper subset of B.

We need to verify whether there exists a normal word, w with satisfies the above properties and finally whether it satisfies α. So as per our guess, our word w should look as follows.

$$w = a_0 w_1 a_1 w_2 \ldots w_{k-1} a_k$$

where the length of $w_1, w_2, \ldots, w_k$ are such that, $w[l_i] = a_i$ for all $i \geq 0$. Thus we need to verify that

$$w, l_i \models \bigwedge_{\beta \in B_i} \mathrm{F}\beta \wedge \bigwedge_{j=0}^{i-1} \bigwedge_{\beta \in B_j} \neg\beta \tag{1}$$

Moreover for all i where $1 \leq i < k$, we have that $|w_i| \leq q$ and no point in w_i is a witness state for any formula. Therefore w_i satisfies the following:

$$w, l_i + 1 \models \mathrm{G} \bigwedge_{j=0}^{i} \bigwedge_{\beta \in B_j} \neg\beta \tag{2}$$

We verify these by iterating through all $i \leq k$ in a descending order. As the base case, we verify Condition (1) for $i = k$. All modalities can be stripped away and verified against the propositions true at this state and the location of the state. We give the following mapping $\nu : SubF(\alpha) \to \{\top, \bot\}$ such that

$$\nu(\mathrm{F}\beta) = \begin{cases} \top & \text{if } \beta \in B_k \\ \bot & \text{otherwise} \end{cases}$$

That is ν maps all future formulas guessed to be true at a_k to be *true* and the rest to be $false$. Using ν, we give a new mapping, $\hat{\nu} : B \to \mathrm{TL}[\mathrm{LEN}]$ as follows. Let $\gamma \in B$. We now replace all future formula in γ by $\{\top, \bot\}$ depending on the ν function. That is, if $\mathrm{F}\beta \in SubF(\gamma)$ then replace $\mathrm{F}\beta$ in γ by $\nu(\mathrm{F}\beta)$. Now we verify if for all $\gamma \in B$, whether $w, l_k \models \hat{\nu}(\gamma)$ or not. Thus we have verified the guesses at $i = k$. Now let us look at an $i < k$. From our iteration, we could assume that for all $j > i$ we have that the guesses are correct. Let us first look at verifying Condition (2). We give the following mapping $\nu : SubF(\alpha) \to \{\top, \bot\}$ such that

$$\nu(\mathrm{F}\beta) = \begin{cases} \top & \text{if } \beta \in \cup_{j>i} B_j \\ \bot & \text{otherwise} \end{cases}$$

Again, we give a new mapping, $\hat{\nu} : B \to \mathrm{TL}[\mathrm{LEN}]$ as was given above. That is by replacing all subformulas by $\top$ or $\bot$ depending on whether $\nu(\mathrm{F}\beta)$ is $\top$ or not. Let

$\gamma = G \bigwedge_{j=0}^{i} \bigwedge_{\beta \in B_j} \neg \hat{\nu}(\beta)$, got by replacing β by $\nu(\beta)$ in the formula in Condition (2). We need to check whether there exists a word, w_{i+1} of length $(l_{i+1} - l_i + 1)$ such that $w[1, l_i]w_{i+1}, l_i + 1 \models \gamma$. From Lemma 5.3.5, we see that this is the *BlockSAT* property, checkable in Π_2^P. We claim that this is equivalent to checking whether $w, l_i + 1 \models \mathrm{G} \bigwedge_{j=0}^{i} \bigwedge_{\beta \in B_j} \neg\beta$. Let us assume not. That is, let β be the smallest formula such that $w, l_i + 1 \models \mathrm{F}\beta$ but $w, l_i + 1 \not\models \hat{\nu}(\beta)$. But this is impossible, since by induction hypothesis $w, l_{i+1} \not\models \mathrm{F}\beta$ and by the Π_2^P machine we know that for all $t \in \mathbb{N}$ such that $l_i < t \leq l_{i+1}$ $w, t \not\models \beta$, since $w, t \not\models \hat{\nu}(\beta)$.

We now need to verify Condition (1). The argument for this is similar to that for $i = k$.

And finally we also need to check that.

$$w, 1 \models \alpha \tag{3}$$

This is again checkable by replacing all subformulas $\mathrm{F}\beta \in SubF(\alpha)$ by $\top$ or $\bot$ depending on whether there exists some j such that $\beta \in B_j$ or not. This gives us a formula in TL[LEN], which can be checked easily.

The algorithm we have described is an NP procedure which uses a Π_2^P oracle and hence is in Σ_3^P. □

5.3.2 *Length modulo counting - Lower bound*

In this section we show that the satisfiability problem for TL[F, LEN] is Σ_3^P-hard, even if we use unary notation and finite word models. We denote by $\beta[p/\phi]$ the formula got by replacing all occurrences of the proposition p by ϕ in the formula β.

Let QBF_3 be the set of all quantified boolean formulas which starts with an existential block of quantifiers followed by a universal block of quantifiers which are then followed by an existential block of quantifiers (that is there are three quantified alternation) Checking whether a QBF_3 formula is true is Σ_3^P-complete. We reduce from evaluation of QBF_3 formulae to satisfiability of our logic.

Theorem 5.3.8. *Satisfiability for* TL[F, LEN] *is hard for* Σ_3^P*, even if unary notation is used for the syntax.*

Proof. Let us take a formula β with three levels of alternation and which starts with an existential block.

$$\beta = \exists x_1, ..., x_k \forall y_1, ..., y_l \exists z_1, ..., z_m B(x_1, ..., x_k, y_1, ..., y_l, z_1, ..., z_m)$$

We now give a satisfiability-preserving TL^{un}[F,LEN] formula $\widehat{\beta}$ such that β is in Σ_3^P-SAT iff $\exists w, (w, 1) \models \widehat{\beta}$.

Take the first l prime numbers $p_1, ..., p_l$. Replace the y_js by ℓ^P_{0,p_j}. We give the formula $\widehat{\beta}$ below. It is a formula over the x and z propositions, got by replacing y_j by ℓ^P_{0,p_j}, for all $j \leq l$.

$$\widehat{\beta} = \mathrm{G}(\ B[y_j/\ell^P_{0,p_j}]) \wedge \mathrm{F}(\wedge^l_{j=1}\ell^P_{0,p_j}) \wedge \bigwedge_{i=1}^{k}(\mathrm{G}x_i \vee \mathrm{G}\neg x_i)$$

Thanks to the prime number theorem we do not have to search too far (By the prime number theorem, asymptotically there are l primes less than $l \log l$ and hence finding them can be done in polynomial time.) for the primes, and primality testing can be done in polynomial time.

We now show that the following claim is true.

$$\beta \text{ is satisfiable } \Leftrightarrow \widehat{\beta} \text{ is satisfiable}$$

For bit vectors $v \in \{0,1\}^k$, $s \in \{0,1\}^l$ let us denote by β^v_s the formula got by removing the outermost existential and universal quantifier in β and replacing x_i by $v(x_i)$, for all $i \in [k]$ and y_j by $s(y_j)$, for all $j \in [l]$, in β. That is

$$\beta^v_s = \beta[x_i/v(x_i)][y_j/s(y_j)]$$

Also for a number $j \in \mathbb{N}$, we denote by $\hat{j}$ the vector $(j \ mod\ p_1 = 0, j \ mod\ p_2 = 0, ..., j \ mod\ p_l = 0)$.

$(\Rightarrow)$: Since β is satisfiable, there exists a $v \in \{0,1\}^k$ such that for all $s \in \{0,1\}^l$ the formula β^v_s is satisfiable. Hence for every such s, there is a vector $Z^s \in \{0,1\}^m$ such that $\beta^v_s[z_i/Z^s(z_i)]$ is valid.

We now construct a word w such that $w, 1 \models \widehat{\beta}$. Let $w = a_1 \ldots a_n$ have the following three properties.

- For every $i \leq m$ and for every $j \leq n$, $z_i \in a_j$ if and only if $Z^{\hat{j}}(z_i) = 1$.
- $n = \prod^l_{j=1} p_j$.
- For every $i \leq k$ and for every $j \leq n$ we set $x_i \in a_j$ if and only if $v(x_i) = 1$.

Note that the above three conditions satisfies the first, second and third subformula of $\widehat{\beta}$ respectively. Hence $w, 1 \models \widehat{\beta}$.

$(\Leftarrow)$: Let $w = a_1 \ldots a_n$ be such that $w, 1 \models \widehat{\beta}$. Then $n \geq \prod^l_{j=1} p_j$, since w satisfies $\mathrm{F}(\wedge^l_{j=1}\ell^P_{0,p_j})$. Now consider vectors $v \in \{0,1\}^k$ and for all vectors $s \in \{0,1\}^l$ the vector $Z^s \in \{0,1\}^m$ such that

- For all $i \leq k$, $v(x_i) = 1 \Leftrightarrow x_i \in a_1$.

- For every $s \in \{0,1\}^l$ we find a number $j \leq n$ such that $\hat{j} = s$. Such a j exists since n is large enough. Then consider the vector Z^s such that $Z^s(z_i) = 1 \Leftrightarrow z_i \in a_j$ for every $i \leq m$.

To show that β is satisfiable, we need to give an assignment to the x_is such that for all assignments to the y_js, there is an assignment to the z_rs which evaluates to true. These are given by the assignments v and Z^ss.

□

5.4 SATISFIABILITY OF TL[DUR] LOGIC

Let us first recall the definition of TL[DUR] logic from the Preliminaries (Chapter 2). This is a past logic closed under boolean combination and which can count propositions modulo a number. Let $\alpha \in$ TL[DUR] over the propositions $\mathcal{P}$. Observe that since α is a past logic, we want to check whether there exist some word $w \in (2^{\mathcal{P}})^*$ such that $w, |w| \models \alpha$. Our idea is as follows. We first list down all the MOD formulas in α. Let

$$MOD(\alpha) = \{\text{MOD}^P_{r,q} p_i \text{ such that it is a subformula of } \alpha\}$$

Let $q = lcm\{q_i \mid \text{MOD}^P_{r,q_i} p \in MOD(\alpha)\}$. Now we guess the number of occurrences of each of the p_is in the word (without the last state). Let these be $n_1, ..., n_l$. The problem is now to check if there exists a word where each of the p_is occur $n_i \ (mod\ q)$ times and whether such a word can be extended by a state to satisfy α. Clearly the second condition has a short proof (a valuation) and hence can be verified easily. But what about the first. Below we give a small certificate for the first condition too. We prove a stronger claim below in the problem $COUNTSAT(PROP)$. We claim that this problem is in NP and hence

---

$COUNTSAT(PROP)$:

Input: Set of Propositions, $\mathcal{P} = \{\alpha_1, \ldots, \alpha_k\}$ and numbers $q, n_1, ..., n_k \in \mathcal{N}$ (numbers given in binary)
Question: There exists a $w \in (2^{\mathcal{P}})^*$ such that $\forall i \leq k : |\{j \mid (w, j) \models \alpha_i\}| \equiv n_i \ (mod\ q)$?
Complexity: *NP-Complete*

---

complete for NP.
See that the size of w can be atmost exponential in the size of the input, since it can be atmost of size q, which is exponential in the size of the input (q is written in binary). Hence guessing w and verifying it wont give an NP algorithm. Let the different states (valuations) of w be $\mathcal{V} = \{v_1, .., v_n\}$. For a set of proposition $\mathcal{P}$ and valuations $\mathcal{V}$ we define a $k \times n$ matrix $A_{\mathcal{P},\mathcal{V}}$ such that $k = |\mathcal{P}|$ and $n = |\mathcal{V}|$.

$$A_{\mathcal{P},\mathcal{V}}(i,j) \quad = \quad \begin{cases} 1 & \text{if } (v_j \models \alpha_i),\ v_j \in \mathcal{V}, \alpha_i \in \mathcal{P} \\ 0 & \text{otherwise} \end{cases}$$

Let $\hat{w}$ be the parikh vector of the word w (this vector counts the number of times each different valuation appears in w). See that $(A\ \hat{w})$ will give the number of positions in w which satisfies the α_is (We drop the subscript from now on). Call the vector $(n_1, ..., n_k)$ by $\vec{b}$. In Equation 4 we give an example matrix A and a vector $\hat{w}$.

$$\begin{array}{c} \\ \alpha_1 \\ \alpha_2 \\ ... \\ ... \\ \alpha_k \end{array} \begin{array}{c} \begin{array}{cccc} \vec{v_1} & \vec{v_2} & \vec{..} \vec{..} & \vec{v_n} \end{array} \\ \begin{pmatrix} 1 & 0 & & 1 \\ 0 & 1 & & 0 \\ ... & ... & & ... \\ 1 & 0 & & 1 \\ 0 & 1 & & 0 \end{pmatrix} \end{array} \begin{array}{c} \vec{\hat{w}} \\ \begin{pmatrix} 3 \\ 5 \\ ... \\ ... \\ 1 \end{pmatrix} \end{array} \equiv_q \begin{array}{c} \vec{b} \\ \begin{pmatrix} n_1 \\ n_2 \\ ... \\ ... \\ n_k \end{pmatrix} \end{array} \tag{4}$$

Therefore what $COUNTSAT(PROP)$ ask is, whether there exists a w such that

$$(A\ \hat{w}) mod\ q = b$$

Now we show a result from linear algebra, which is useful in proving the NP upper bound for $COUNTSAT(PROP)$. Consider a $k \times n$ integer matrix, A where $n > k$. The following operations on a matrix are called elementary column operations: exchanging two columns, multiplying a column by -1, and adding an integral multiple of one column to another column. We now claim the following.

Lemma 5.4.1. *[Sch98] Any $k \times n$ integer matrix A (where $n > k$) can be transformed into a matrix with $n - k$ zero columns (0 vector) by applying a series of elementary column operations.*

Proof. By induction on the number of rows, k. See that if number of rows is 1, the claim is true. Now Consider a matrix with k rows. Assume by our induction hypothesis we are in a position $\begin{bmatrix} B & 0 \\ C & D \end{bmatrix}$, where B is a square matrix. With elementary column operations we can modify D such that its first row$(d_1, ..., d_l)$ is non-negative and also such that the $\sum d_i$ is as small as possible. We assume $d_1 \geq d_2 \geq ... \geq d_l$ (if not interchange columns). We claim that $d_2 = 0$. Assume not. Then we can subtract this column with the first column(which contains d_1) making the $\sum d_i$ even smaller, a contradiction. Hence it now follows that $d_2 = d_3 = ... = d_k = 0$ and we have solved the problem for a larger matrix (the matrix B with this new row till d_1). Repeating this procedure we can convert the full matrix into the form we wanted. □

We now prove a small model property for the problem $COUNTSAT(PROP)$. We can also view this as a polynomial sized certificate, which we can verify in polynomial time making the problem in NP.

Lemma 5.4.2. *Let $\mathcal{P}$ be a set of Propositions of size k. Then we claim*
$\langle \mathcal{P}, q, n_1, ..., n_k \rangle \in COUNTSAT(PROP)$ iff
$\exists \mathcal{V} \subset 2^{\mathcal{P}}$, where $|\mathcal{V}| = k$ and $\exists \vec{v} \in \{0, ..., q\}^k$ such that $(A_{\mathcal{P},\mathcal{V}}\ \vec{v}) mod\ q = b$

Proof. $(\Leftarrow)$: This is trivial.
$(\Rightarrow)$: Let there exists a w which satisfies the conditions of the problem. Therefore there exists a matrix, B as given in Equation 4. See that $(B\hat{w})mod\ q = b$. Unfortunately the matrix might have more columns than k. From Lemma 5.4.1. it follows that there exists k columns(called basis vectors) such that any column in B can be represented by an integral combination of these basis vectors. Call this new matrix A formed by taking these k basis vectors. Also there exists a parikh vector say v such that $A\ v \equiv_q B\hat{w} \equiv_q b$. If any component of v is greater than q, then just take modulo q of that number. This proves the claim. □

Using the above theorem, it is easy to give an NP upper bound for $COUNTSAT(PROP)$.

Theorem 5.4.3. *$COUNTSAT(PROP)$ is* NP*-complete.*

Proof. Lemma 5.4.2 gives us a polynomial sized certificate to be verified. We guess the matrix A, and the vector v and we check whether $A\ v \equiv_q b$. The latter verification can be done in polynomial time. Lower bound is because boolean satisfiability is in NP. □

It is now easy to show that satisfiability of TL[DUR] is in NP. The proof comes from the arguments we made above showing $COUNTSAT(PROP) \in$ NP.

Theorem 5.4.4. *Satisfiability of* TL*[*DUR*] is* NP*-complete.*

5.5 DISCUSSION

In this chapter we looked at satisfiability and model checking problems for various extensions of LTL. The extensions we looked at were all under the regular framework. We had LTL extended with modulo counting and group extensions. The previous chapter 4 looked at the expressiveness of these logics. This chapter shows that these extended logics can very well be used for verification purposes, since they have the space complexity exactly like LTL. This seems to have escaped the notice of verification researchers until now.

An interesting question to ask is: Are there other families of automata, where a "standard" enumeration of their states and transitions can be represented in logarithmic notation, and for which the PSPACE bound will continue to hold? Identifying these families will help us in using them for verification purposes.

A patent weakness of our approach is that LTLGRP specifications are far from perspicuous, but we look to demonstrate an idea, and it will take specification examples from practice to provide useful patterns for the more expressive logic.

We also observed that when LTL is extended with modulo counting, it does not matter if the specification of the moduli is in succinct notation or not. That is even for a very weak extension of LTL, that is TL[F,MOD(2)] we get PSPACE-hardness. More generally this holds for computation within a finite symmetric group.

Another point to note is that our PSPACE hardness proof required introducing polynomial number of propositions. Will the lower bound continue to hold in the case of constant number of propositions? We believe not.

Open Problem 5.5.1. *A tight upper and lower bound for* TL[F,MOD(2)] *with constant number of propositions is not known. Currently we have an upper bound of* PSPACE *and an* NP *lower bound.*

If this is the case then it contrasts with TL[X,F], since it is PSPACE-hard for constant number of propositions. We bring to your notice a result by Weis and Immerman [WI09], where it was shown that the satisfiability of $FO^2[<]$ is NEXPTIME-complete but with constant number of propositions it is NP-complete.

Consider the logic with only the modulo counting operator but no temporal logic modality. We do not know the complexity of satisfiability of this logic. The logic TL[DUR], looks at the special case when modulo counting is allowed only over propositions.

Open Problem 5.5.2. *The complexity of* TL[MOD] *is not known. We know that it lies in between* NP *and* PSPACE.

6

FO[<] SATISFIABILITY

6.1 INTRODUCTION

Since satisfiability of FO[<] itself is non-elementary over words, it is of interest to study the counting quantifiers in a weaker framework, such as the two-variable framework.

The first contribution of this chapter is to show that this upper bound extends to the slightly stronger $FO^2[<, succ, \equiv]$, and that the lower bound holds even for $FO^2[\equiv]$ (FO^2 with only the modulo predicates or $FO^2[\equiv]$, that is no less than or successor relation) for a constant alphabet size.

The second contribution of this chapter is to show that the satisfiability of $FO^2\text{MOD}[<, succ]$ is in EXPSPACE. We also show a corresponding lower bound for the weaker logic $FO^2\text{MOD}(2)[<, \equiv]$. Our upper bound results assume that the integers in modulo quantifiers and modulo predicates are in binary, whereas our lower bound results assume they are in unary. Thus the complexity does not depend on the representation of integers. We also extend our results to show that the same upper bound holds for computation over finite groups, which is the present decidability frontier. Again our upper bound assume succinct descriptions of groups using generators rather than group elements.

6.2 UPPER BOUNDS VIA LINEAR TEMPORAL LOGIC

In this section we show the satisfiability of modulo counting logics. It is easy to observe that there exists a linear time translation from TL[F,P,X,Y,GRP] to $FO^2\text{GRP}[<, succ]$. There is an *exponential time* translation in the reverse direction. We restate Theorem 4.3.1 found in Chapter 5 below.

Lemma 6.2.1. *For an $FO^2\text{MOD}[<, succ]$ ($FO^2[<, succ, \equiv]$, respectively) formula α of quantifier depth d and size n there exists a TL[X,Y,F,P,MOD] (TL[X,Y,F,P,LEN], resp.) formula α' of modality depth $2d$ and size at most $O(2^n)$, such that α and α' accept the same set of word models. Moreover this translation can be done in exponential time.*

Combining Lemma 6.2.1 with Corollary 5.2.2 that TL[X,Y,F,P,MOD] satisfiability is in PSPACE we get that:

Theorem 6.2.2. *FO^2MOD[<, succ] satisfiability is in* EXPSPACE.

Proof. Given a sentence $\alpha \in \text{FO}^2\text{MOD}[<, succ]$ then using Lemma 6.2.1 we can convert it to an TL[X,Y,F,P,MOD] formula of exponential size. Theorem 5.2.2 then gives us that satisfiability is in EXPSPACE. □

A similar result can be given for the extended logic $\text{FO}^2\text{GRP}[<, succ]$ by combining Lemma 6.2.1 with Theorem 5.2.2 that TL[F,P,X,Y,GRP] satisfiability is in PSPACE.

Theorem 6.2.3. *FO^2GRP[<, succ] satisfiability is in* EXPSPACE.

For $\text{FO}^2[<, succ, \equiv]$ we can do better. We next show that a satisfiable TL[X,Y,F,P,LEN] formula α has a model which is polynomial in $\text{lcm}(\alpha)$ (where $\text{lcm}(\alpha) = \text{lcm}\{q \mid x \equiv_q r$ is a subformula in $\alpha\}$), but which is exponential in the modality depth and number of propositions. Observe that the size of the formula is irrelevant for the model size.
The following definitions are borrowed from Etessami et al [EVW02]. They had defined these for UTL. We extend them to TL[X,Y,F,P,LEN].

Definition 6.2.4. *The* depth *of a formula $\alpha \in$ TL[X,Y,F,P,LEN] is (k,k') if its* {F, P} *depth is k and its* {X, Y} *depth is k'.*

Definition 6.2.5. *For a word w and a position i, the* (k,k')-type *of i in w, denoted by $\tau_{(k,k')}(w,i)$ is the set of all* TL[X,Y,F,P,LEN] *formulas of depth (k,k') that hold in w at i. That is*

$$\tau_{(k,k')}(w,i) = \{\phi \mid w,i \models \phi, \phi \in \text{TL}[\text{X,Y,F,P,LEN}], \text{ depth of } \phi \text{ is } (k,k')\}$$

Let us denote by $\mathcal{P}$ the set of propositions and let $p = |\mathcal{P}|$, be the number of propositions.

The following lemmas (Lemma 6.2.6 and 6.2.7) say that the question of satisfiability of a formula in TL[X,Y,F,P,LEN], can be reduced to counting the number of distinct (k,k')-types possible in a word. Let $T(k,k')$ be the maximum number of distinct (k,k')-types possible in a word.

Let α be a TL[X,Y,F,P,LEN] formula over the alphabet $\mathcal{P}$. Observe that if α is an UTL formula we can use the NEXPTIME algorithm given in [EVW02] to check whether α is satisfiable. Hence we assume that α is not an UTL formula and therefore $lcm(\alpha)$ is defined and let it be equal to q. Our first aim is to understand when do two words at two positions have the same (k,k')-type.

Lemma 6.2.6. *For two words w, w' and positions i, i', where $1 + k' \leq i \leq |w| - k'$ and $1 + k' \leq i' \leq |w'| - k'$*

$$\tau_{(0,k')}(w,i) = \tau_{(0,k')}(w',i') \text{ iff } w[i-k', i+k'] = w'[i'-k', i'+k'] \text{ and } i \equiv_q i'$$

Proof. A depth $(0,k')$ formula can only look at the position modulo q and the letters in the positions at a distance of k' to the left or right of the current position. □

The above lemma looked at the base case, when the {F, P} modalities are not present in the formula. The following lemma captures the general case.

Lemma 6.2.7. *For two words w, w' and positions i, i', where $i \leq |w|$ and $i' \leq |w'|$*

$$\tau_{(k+1,k')}(w,i) = \tau_{(k+1,k')}(w',i') \text{ iff } \begin{cases} \tau_{(0,k')}(w,i) = \tau_{(0,k')}(w',i') \text{ and} \\ \{\tau_{(k,k')}(w,j) \mid j < i\} = \{\tau_{(k,k')}(w',j) \mid j < i'\} \text{ and} \\ \{\tau_{(k,k')}(w,j) \mid j > i\} = \{\tau_{(k,k')}(w',j) \mid j > i'\} \end{cases}$$

Proof. Every TL[X,Y,F,P,LEN] formula can be written in a "normal form", where all the X, Y modalities are pushed inside the {F, P} modalities. This does not change the depth of the formula [EVW02].
($\Rightarrow$) : Consider an α of the form Xϕ such that $\alpha \in \tau_{(0,k')}(w,i)$. Since the left hand side is true, we have that α is also in $\tau_{(0,k')}(w',i')$. Similarly $\alpha \notin \tau_{(0,k')}(w,i)$ implies that $\alpha \notin \tau_{(0,k')}(w',i')$. So let us consider an α of the form Fϕ (a similar analysis can be given for $\alpha = $ Pϕ). Let $\phi \in \tau_{(k,k')}(w,j)$, for a $j > i$. Then clearly we have $w, j \models \phi$ and therefore $w, i \models \alpha$. Since the left hand side is true, we get that $w', i' \models \alpha$ and hence there exists a $j' > i'$ such that $w', j' \models \phi$ and hence $\phi \in \tau_{(k,k')}(w',j')$ holds. A similar argument can be used to show that if $\phi \notin \tau_{(k,k')}(w,j)$ implies $\phi \notin \tau_{(k,k')}(w',j')$.
($\Leftarrow$) : Let α be a formula in the normal form as stated above. Let $\alpha \in \tau_{(k+1,k')}(w,i)$ such that $\alpha \in \tau_{(0,k')}(w,i)$. But then $\alpha \in \tau_{(0,k')}(w',i')$. So let us assume that α is of the form Fϕ (the case when $\alpha = $ Pϕ will be analogous). Then clearly there exists a $j > i$ such that $w, j \models \phi$, which implies that $\phi \in \tau_{(k,k')}(w,j)$. This (since the left hand side is assumed to be true) implies that $\phi \in \tau_{(k,k')}(w',j')$ for a $j' > i'$. Hence the claim holds. □

We now show how to get a *small model property* for TL[X,Y,F,P,LEN]. The proof of the following Lemma follows the proof of Lemma 3 in [EVW02].

Lemma 6.2.8. *Let $w = u_1 \dots u_i \dots u_{i'} \dots u_n$ be a word and $i, i' \geq 1$ such that $\tau_{(k,k')}(w,i) = \tau_{(k,k')}(w,i')$ Let $w' = u_1 \dots u_i u_{i'+1} \dots u_n$. Then we have that*

1. *for all $j \leq i$ $\left(\tau_{(k,k')}(w,j) = \tau_{(k,k')}(w',j)\right)$*
2. *for all $j \geq i'$ $\left(\tau_{(k,k')}(w,j) = \tau_{(k,k')}(w', j - i' + i)\right)$*

Proof. We prove by induction on k. Let $k = 0$. Since for all $i \leq |w'|$, $w[i-k', i+k'] = w'[i-k', i+k']$, the claim holds for the base case.

Now, let us assume that the claim is true for all numbers less than or equal to k and let $\tau_{(k+1,k')}(w,i) = \tau_{(k+1,k')}(w,i')$. Then by Lemma 6.2.7 $\{\tau_{(k,k')}(w,j) \mid j < i\} = \{\tau_{(k,k')}(w,j) \mid j < i'\}$ and therefore

$$\{\tau_{(k,k')}(w,j) \mid i < j < i'\} \subseteq \{\tau_{(k,k')}(w,j) \mid j < i\} \tag{5}$$

Similarly Lemma 6.2.7 gives us that

$$\{\tau_{(k,k')}(w,j) \mid i < j < i'\} \subseteq \{\tau_{(k,k')}(w,j) \mid j > i'\} \tag{6}$$

We now define a mapping

$$\nu(j) = \begin{cases} j & \text{if } j \leq i \\ j - i + i' & \text{otherwise} \end{cases}$$

From the inductive hypothesis, we know that $\tau_{(k,k')}(w,\nu(j)) = \tau_{(k,k')}(w',j)$, for all $j \leq |w'|$. Therefore we have $\{\tau_{(k,k')}(w',j) \mid j < i\} = \{\tau_{(k,k')}(w,\nu(j)) \mid j < i\}$ and using (5) we get $\{\tau_{(k,k')}(w,\nu(j)) \mid j < i\} = \{\tau_{(k,k')}(w,j) \mid j < i'\}$. Similarly we can show $\{\tau_{(k,k')}(w',j) \mid j > i\} = \{\tau_{(k,k')}(w,j) \mid j > i'\}$. The claim now follows from Lemma 6.2.7. □

The above lemmas showed that if the two positions of a model have the same (k,k') type, then we can construct a smaller model. This shows that the number of distinct (k,k') types determine the smallest model of a formula. The following lemma counts the number of distinct (k,k') types true in a word. Recall that $T(k,k')$ is the number of distinct (k,k') types possible in any model.

Lemma 6.2.9. *Let w be a word. Then the $(k+1,k')$-type at position i in w is uniquely given by the $(0,k')$-type at i, the (k,k')-types that occur to its right and the (k,k')-types that occur to its left. In particular $T(k+1,k') \leq q^{2(p+1)(2k'+1)(k+1)}$.*

Proof. For $k = 0$, Lemma 6.2.6 gives us that $T(0,k') \leq q2^{(p+1)\cdot(2k'+1)}$. Consider a word w with $T(k,k')$ different types. At any position i in the word consider the following pair $\Big(\{\tau_{(k,k')}(w,j) \mid j \leq i\}, \{\tau_{(k,k')}(w,j) \mid j > i\}\Big)$. The number of such distinct pairs is equal to $2T(k,k')+1$ (since the left hand side is an increasing set). Therefore we can now write $T(k+1,k') = q.(2T(k,k')+1)T(0,k') \leq (2qT(0,k'))^{(k+1)}$. This proves the claim. □

Finally we are in a position to show a small model property for TL[X,Y,F,P,LEN].

Lemma 6.2.10. *Every satisfiable* TL[X,Y,F,P,LEN] *formula α of depth (k,k') has a model of size less than $T(k,k')+1$.*

Proof. Let $w = u_1 \cdots u_n$ be a model for α and let $n \geq T(k,k')+1$. Then there exists positions i, i' such that they have the same type. From Lemma 6.2.8 we can give a smaller w' such that $\tau_{(k,k')}(w,1) = \tau_{(k,k')}(w',1)$. □

The above small model property for TL[X,Y,F,P,LEN] gives us that:

Theorem 6.2.11. *$FO^2[<, succ, \equiv]$ satisfiability is in* NEXPTIME.

Proof. Lemma 6.2.1 shows that for every $\mathrm{FO}^2[<, succ, \equiv]$ formula α, there exists an TL[X,Y,F,P,LEN] formula α' such that α and α' have the same set of models. Moreover, if the quantifier depth of α is d, then the operator depth of α' is $2d$. Lemma 6.2.10 shows that every satisfying TL[X,Y,F,P,LEN] formula α' of operator depth $2d$ has a satisfying model of size $s = O(lcm(\alpha)^{4d^2} 2^{4pd^2})$. A NEXPTIME machine can guess this model and verify it in time $s^2 \times |\alpha|$. □

6.3 LOWER BOUNDS VIA TILING PROBLEMS

We now give the lower bound for these logics. While showing lower bound results, as already explained, we will assume that all the "numbers" in the formulas will be written in unary notation. We first show that FO^2MOD[<] where the modulo numbers are written in unary notation is EXPSPACE hard. Then we show that $\mathrm{FO}^2[\equiv]$, where the numbers for the modulo predicates are written in unary, is NEXPTIME hard. Moreover, we assume that the formulas have an alphabet size of 2. Note that the satisfiability problem for the logic $\mathrm{FO}^2[<]$ for a fixed alphabet is NP-complete [WI09].

The lower bound results in this section are shown by reducing from Tiling problems. We define the required Tiling problems now.

6.3.1 *Tiling problems*

A *tiling system* [WTC61] is a tuple $S = (T, Rt, Dn)$, where T is a finite set of tiles, $Rt \subseteq T \times T$ and $Dn \subseteq T \times T$ are, respectively, the right (horizontal) and down (vertical) adjacency relations (We will use these to define a constant size alphabet in a reduction). A *tiling problem* is the tuple $(S, n, top_1, ..., top_n, bot)$, where $n \in \mathbb{N}$ and is given in unary and $top_1, ..., top_n, bot \in T$. A *tiling* of an $m \times k$ grid $R \subseteq \mathbb{N}^2$ is a mapping $\tau : R \to T$ respecting the right and down relations, that is, whenever $(i, j+1)$ or $(i+1, j)$ is in R, we have $Rt(\tau(i,j), \tau(i,j+1))$ or $Dn(\tau(i,j), \tau(i+1,j))$, as the case may be.

We give below two versions of the tiling problem $(S, n, top_1, ..., top_n, bot)$ corresponding to EXPSPACE and NEXPTIME Turing machines respectively. Pan and Vardi [PV06] give a similar argument for an NP-complete problem.

RECTANGLE TILING PROBLEM Given a tiling problem $(S, n, top_1, \ldots, top_n, bot)$, the Rectangle tiling problem asks, does there exist an m and a tiling of an $m \times 2^n$ grid such that the first n tiles in the top row is $top_1, ..., top_n$ in order and there exists a tile bot in the bottom row?

Proposition 6.3.1. *There exists a tiling system $S = (T, Rt, Dn)$, such that its Rectangle tiling problem $(S, n, top_1, ..., top_n, bot)$ is* EXPSPACE*-complete.*

SQUARE TILING PROBLEM Given a tiling problem $(S, n, top_1, \ldots, top_n, bot)$, the Square tiling problem asks, does there exist a tiling of an $m \times m$ grid, where m is the product of the first n primes, such that the first n tiles in the top row is $top_1, ..., top_n$ in order and there exists a tile *bot* in the bottom row?

Proposition 6.3.2. *There exists a tiling system $S = (T, Rt, Dn)$, such that its Square tiling problem $(S, n, top_1, ..., top_n, bot)$ is* NEXPTIME*-complete.*

6.3.2 *Modulo counting is* EXPSPACE*-hard*

We show that satisfiability of FO^{2}MOD[<] is EXPSPACE-hard by reducing from the EXPSPACE-complete Rectangle tiling problem. The following lemma shows that $x \equiv y(\mod 2^n)$ is definable by an FO^{2}MOD[<] formula, which allows us to specify the "down" relation in a tiling system.

We denote by $Odd\ y\ \alpha$ the formula $\exists^{0,2}\ y\ \alpha$.

Lemma 6.3.3. *There exists an algorithm which given an $n \in \mathbb{N}$ can output a formula, $\chi_n(x, y) \in FO^2$MOD*[<] such that $\chi_n(x, y)$ is of size $O(n)$ and quantifier depth 2 and such that $\chi_n(x, y)$ is true iff $x \equiv y(\mod 2^n)$.*

Proof. For all $i \leq n$, we give formulas $\psi_i(x)$ such that $\psi_i(x)$ is true iff the i^{th} least significant bit (lsb) of x is 1. $\psi_1(x)$ is true if x is odd and is given by the formula $Odd\ y(y \leq x)$. For all $i \geq 2$:

$$\psi_i(x) := Odd\ y(y < x \wedge \exists^{(2^{i-1},2^i)} x(x \leq y))$$

Let $C(x)$ be the number of positions y which satisfy the conditions $y < x$ and $y \equiv 2^i - 1\ (mod\ 2^i)$. That is

$$C(x) = |\{y < x \mid y \equiv 2^i - 1\ (\mod 2^i)\}|$$

. We show by induction on $C(x)$ that

$$\psi_i(x) \text{ is true } \Leftrightarrow i^{th} \text{ lsb of } x = 1$$

When $C(x) = 0$, we know that $x < 2^i - 1$. This means the i^{th} lsb is 0 and therefore the claim is true for the base case. For the induction step, assume the claim to be true for all $x \leq z$. Let $C(z) = k$ and $C(z+1) = k+1$. Therefore by IH, we know that for all $x \leq z$, $\psi_i(x)$ is true iff the i^{th} least significant bit of x is 1. Since $C(z) = k$ and $C(z+1) = k+1$, we have $z \equiv 2^i - 1(\mod 2^i)$. This implies that if we add an 1 to z the i^{th} bit toggles. The i^{th} bit will be the same from z to $z + 2^i - 1$. Thus for all numbers $y \in [z, z + 2^i - 1]$, we have $\psi_i(y)$ is true iff i^{th} bit is 1.

Now $\chi_n(x, y) := \bigwedge_{i=1}^{n}(\psi_i(x) \Leftrightarrow \psi_i(y))$. The size of $\chi_n(x, y)$ is $O(n)$. □

Theorem 6.3.4. *The satisfiability problem for FO^{2}MOD[<] is* EXPSPACE*-hard even for a constant alphabet (and constant quantifier depth).*

Proof. Our reduction is from the EXPSPACE-complete Rectangle tiling problem $I = (S, n, top_1, ..., top_n, bot)$. Here $S = (T, Rt, Dn)$, where $T = \{T_1, ..., T_t\}$. We take two more copies of T, $T^{Dn} = \{T_i^{Dn} \mid T_i \in T\}$ and $T^{Rt} = \{T_i^{Rt} \mid T_i \in T\}$.

We give a polynomial time reduction, which when given an instance of the problem I will output a formula ψ_I over the alphabet $\Sigma = T \cup T^{Dn} \cup T^{Rt}$ such that there exists a tiling for I iff ψ_I is satisfiable. For a tiling τ, we associate a word model $M_\tau \in \mathcal{P}(\Sigma)^*$ such that τ is a tiling for I iff $M_\tau \models \psi_I$. We denote by $M_\tau(i, j)$ the letters at the $(i-1)2^n + j^{th}$ position in M_τ. We will ensure that M_τ will satisfy the property $\tau(i, j) = T_k \Leftrightarrow T_k \in M_\tau(i, j)$.

The formula ψ_I is a conjunction of the formulas $\psi_{init}, \psi_{final}, \psi_{next}, \psi_{constraints}$ describing the initial input configuration, the final configuration, the next move, and the constraints respectively. The input configuration says that the first row contains the tiles $top_1, \cdots, top_n$. The formula ψ_{init} is the conjunction of $\alpha_1, ..., \alpha_n$, where α_i says that the i^{th} cell in the first row contains the tile top_i. This is encoded by saying that the first location x which satisfies $x \equiv i \ (mod\ 2^n)$ is the i^{th} cell in the first row.

$$\alpha_i := \forall x \ ((x \equiv i \ (\mod 2^n) \wedge \forall y < x \neg(y \equiv i \ (\mod 2^n))) \Rightarrow top_i(x))$$

Claim 6.3.5. *The formula $\psi_{init} = \bigwedge_i \alpha_i$ is such that*

$$M_\tau \models \psi_{init} \Leftrightarrow \text{for all } i \leq n,\ M_\tau(1, i) = top_i$$

Proof. α_i is satisfied only if $M_\tau(1, i) = top_i$. □

The final configuration says that there exists a tile *bot* in some cell. This is given by the formula, $\psi_{final} := \exists y\, bot(y)$. Finally $\psi_{constraints}$ says that only one tile is true in a position and can be given by an $FO^2[<]$ formula.

The non-trivial part is to show that the down and right relations are respected. The formula $\psi_{next} := \psi_{down} \wedge \psi_{right}$, ensure that the relations *Dn* and *Rt* are respected. We first explain in some detail how the down constraint is respected.

Let us say tile $T_{l'}$ is true at $M_\tau(i, j)$ and let us assume that there is only one tile T_l such that $Dn(T_{l'}, T_l)$ is true. Hence we need to ensure that $T_l \in M_\tau(i+1, j)$. The idea is to count modulo 2, the number of occurrences of T_l^{Dn} in all cells above i and in the same column. That is we count the size of the set $\{k \mid T_l^{Dn} \in M_\tau(k, j),\ k < i\}$. If this count is even then we force T_l^{Dn} to be true at $M_\tau(i, j)$. Otherwise we force T_l^{Dn} to be false at $M_\tau(i, j)$. This will ensure that the count of $\{k \mid T_l^{Dn} \in M_\tau(k, j),\ k \leq i\}$ to be odd. Similarly, for all other tiles $T_s^{Dn} \neq T_l^{Dn}$ we ensure the count $\{k \mid T_s^{Dn} \in M_\tau(k, j),\ k \leq i\}$ to be even. We will preserve this invariant at every cell. Hence the tile at $(i+1, j)$ can now be determined by looking at the count of $\{k \mid T_l^{Dn} \in M_\tau(k, j), k < i+1\}$ for every tile T_l and setting that tile whose count is odd. That is for all i, j we ensure that ($\exists! l$ stands for unique l)

$$T_{l'} \in M_\tau(i, j) \Rightarrow \exists! l \text{ such that } |\{k \mid T_l^{Dn} \in M_\tau(k, j), k \leq i\}| \equiv_2 1 \Leftrightarrow T_l \in M_\tau(i+1, j)$$

The following formula $\psi_1(x)$ says that if you see an odd number of occurrences of the letter T_l^{Dn} in the current column strictly above it, then we set letter T_l to be true at x. This ensures that $T_l \in M(i+1, j)$.

$$\psi_1(x) := \bigwedge_{l=1}^{t} \Big(Odd\ y(T_l^{Dn}(y) \wedge y < x \wedge \chi_n(x,y)\Big) \Rightarrow T_l(x)$$

The following formula ψ_2 says that if a position x contains the tile $T_{l'}$ and the position $(y = 2^n + x)$ contains the tile T_l (that is the cell in the same column, but exactly one row below), then we set the count of the letter T_l^{Dn} above and in the same column as x to be odd. This implies setting T_l^{Dn} to be true or not at x depending on the count above x and in the same column.

$$\psi_2(x) := \bigwedge_{l'=1}^{t} (T_{l'}(x) \Rightarrow \bigvee_{(T_{l'},T_l)\in Dn} (\phi_l(x) \wedge \bigwedge_{j\neq l} \neg\phi_j(x)))$$

Here for all $s \leq t$ we define $\phi_s(x)$ to stand for the following formula, which forces the count of T_s^{Dn} to be even or odd depending on whether $T_s^{Dn} \in M_\tau(i+1, j)$ or not.

$$\phi_s(x) := Odd\ y(T_s^{Dn}(y) \wedge (y \leq x) \wedge \chi_n(x,y))$$

We finally write $\psi_{down} = \forall x\ \psi_1(x) \wedge \psi_2(x)$. ψ_2 enables the count of a particular letter T_l^{Dn} to odd and ψ_1 sets the tile to T_l if the count is odd for T_l^{Dn}.

Claim 6.3.6.

$M_\tau \models \psi_{down} \Rightarrow$ *for all* $i, j, \exists l, l' \in [t]$ $(T_l \in M_\tau(i+1, j)$ *and* $T_{l'} \in M_\tau(i, j))$ *and* $Dn(T_{l'}, T_l)$

Proof. Let us fix an i, j. We show that $M_\tau \models \psi_{down} \Rightarrow$

$$T_{l'} \in M_\tau(i, j) \Rightarrow \exists! l \text{ such that } |\{k \mid T_l^{Dn} \in M_\tau(k, j), k \leq i\}| \equiv_2 1 \Leftrightarrow T_l \in M_\tau(i+1, j)$$

Since $T_{l'} \in M_\tau(i, j)$, and $M_\tau \models \psi_2(i)$ we have that there exists some T_l such that $\phi_l(i)$ is true and for all $s \neq l$, we have $\phi_s(i)$ to be false. This ensures that the first implication is true. Since $\phi_l(i)$ is true, and $M_\tau \models \psi_1(i+1)$ we have the second implication to also hold. □

A similar formula ψ_{right} using the letters T^{Rt} can be written for checking whether the *Right* relations are respected.

Claim 6.3.7.

$$M_\tau \models \psi_I \Leftrightarrow \tau \text{ is a tiling for } I$$

Proof. $(\Rightarrow)$: We show that if there exists a model $M \models \psi_I$ then we can give a valid tiling τ for I. So let $M \models \psi_I$. Our tiling τ is defined as follows. For all i, j

$$\tau(i, j) = M(i, j) \cap T$$

Since $M \models \psi_{constraints}$ we can observe that $\tau(i,j)$ consists of exactly one tile, for all i,j. We have that M satisfy ψ_{init} and hence from Claim 6.3.5 we know that the first row of M is according to the input instance I. Now from Claim 6.3.6 we know that the down relations are respected in M. Similarly ψ_{right} will ensure that the right relations are respected. Finally we have that $M \models \psi_{final}$ and hence we know that there exists the tile type bot. Thus the τ got from model M is a valid tiling for I.
$(\Leftarrow)$: It is easy to observe that we can construct a model M_τ such that $M_\tau \models \psi_I$. □

This completes the proof. □

In the above theorem the numbers in the formulas $\chi_n(x,y)$ were written in binary notation.

Corollary 6.3.8. *The satisfiability problem for $FO^2\text{MOD}(2)[<,\equiv]$ is* EXPSPACE*-hard even when the numbers are in unary notation.*

Proof. Let $p_1, ..., p_n$ be the first n primes and m be their product. Clearly $m \geq 2^n$. By the prime number theorem, asymptotically there are n primes within the first $n \log n$ numbers and hence one can generate the first n primes in time polynomial in n. Next we make use of the Chinese remainder theorem to generate a formula for $x \equiv y(\mod m)$ (but of size $O(n^4)$), because all numbers less than or equal to m (and hence $\leq 2^n$) can be uniquely represented by their remainders modulo the primes $p_1, ..., p_n$. Now we follow the proof of Theorem 6.3.4. □

6.3.3 *Modulo predicates is* NEXPTIME*-hard*

We now show that $FO^2[\equiv]$ is NEXPTIME-hard even for a constant alphabet, as opposed to $FO^2[<]$ being NP-complete [WI09].

Theorem 6.3.9. *$FO^2[\equiv]$ satisfiability is* NEXPTIME*-hard even for constant alphabet size.*

Proof. We reduce from the NEXPTIME-complete Square tiling problem defined earlier. We introduce $2n$ distinct primes, $p_1, ..., p_n$ (for encoding row index), and $q_1, ..., q_n$(for encoding column index). These primes can encode any cell (i,j). One can now write a formula $\alpha_{down}(x)$ which ensures that there exists a y such that the row index of y is one more than that of x and the column index of x and y are the same. The formula can also specify that y should satisfy the down constraints. Similarly one can write a formula to force the right constraints. It is easy to write the initial and final conditions. □

6.4 DISCUSSION

In this chapter we looked at the satisfiability of various counting extensions of first order logic with two variables over words. The counting extensions we looked at are modulo

counting quantifiers, modulo counting predicates. We also looked at extending the logic $FO^2[<]$ by group quantifiers.

The following table 5 gives the status of the satisfiability problem for various extensions of first order logic over words. The results of this chapter are given in parentheses.

Logic	$[<, succ]$	$[<]$
FO^2(increasing alphabet)	NEXPTIME[EVW02]	NEXPTIME-hard [EVW02]
FO^2	NEXPTIME-complete [WI09]	NP-complete [WI09]
$FO^2[\equiv]$	**NEXPTIME**	**NEXPTIME-hard**
FO^2MOD	**EXPSPACE**	**EXPSPACE-hard**
FO^2GRP	**EXPSPACE**	**EXPSPACE-hard**

Table 5: Satisfiability of various two variable logics (all logics other than the first entry assumes increasing alphabet for upper bound and a fixed alphabet for lower bound).

An interesting question would be to identify the exact complexity of the logic FO^2MOD$(2)[<]$. This logic is allowed to use only modulo counting quantifiers over the number 2. No modulo predicates are allowed.

Open Problem 6.4.1. *The complexity of satisfiability of FO^2MOD$(2)[<]$ is known to be between* NP *and* PSPACE. *The exact complexity is not known.*

Part III

EXTENSIONS TO FIRST ORDER LOGIC WITH ADDITION

7

SURVEY ON ADDITION RELATION

In this part we look at the logic FO[<, +] extended with various regular quantifiers. We first look at expressive power of the various logics, especially on proving lower bounds for these logics. Then we look at satisfiability questions.

7.1 EXPRESSIVENESS

Non expressibility results for various logics which uses addition and a variety of quantifiers has been considered earlier. Let us consider the following language, for an $m \in \mathbb{N}$.

$$\mathcal{L}_m = \{w \in \{a,b\}^* \mid |w|_a \equiv (0 \ mod \ m)\}$$

The earliest result was by Lynch [Lyn82] who showed that FO[<, +] cannot count modulo any number.

Theorem 7.1.1. *[Lyn82] For all $m > 1$, $\mathcal{L}_m$ is not definable in FO[<, +].*

Niwiński and Stolboushkin [SN97] looked at numerical predicates of the form $y = px$ and showed that $y = 3x$ is not definable by $y = 2x$ and $<$ relation. Nurmonen [Nur00] extend this result to the case of counting quantifiers. He showed that FOMOD(2)[<, $y = 2x$] cannot define the relation $y = 3x$. Finally Roy and Straubing [RS07] gave lower bound results in the presence of addition and modulo counting quantifiers.

Theorem 7.1.2. *Let $m, n \in \mathbb{N}$ be such that m has a prime factor that does not divide n.*
[SN97] Then $\mathcal{L}_m$ is not definable in FO[<, $y = nx$].
[Nur00] Then $\mathcal{L}_m$ is not definable in FOMOD(n)[<, $y = nx$].
[RS07] Then $\mathcal{L}_m$ is not definable in FOMOD(n)[<, +].

Results by Ruhl [Ruh99a], Lautemann et.al. [LMSV01], Lange [Lan04a], Schweikardt [Sch05], all show the limited expressive power of addition in the presence of majority quantifiers.

Theorem 7.1.3. *[Ruh99a, LMSV01, Lan04a, Sch05] The addition relation is expressible in* MAJ*[<].*
The multiplication relation is not definable in MAJ*[<, +].*

This in effect shows that MAJ[<, +] is a strict subset of the computational complexity class TC^0. What do we know about the regular languages expressible in MAJ[<, +]? Behle, Krebs and Reifferscheid [BKR09b, BKR09a] tries to answer this question. They show that word problems over non-solvable groups are not definable in the two variable fragment of MAJ[<, +].

Theorem 7.1.4. *[BKR09b, BKR09a] Consider a word problem L, over a non-solvable group. Then L is not definable in the two variable fragment of* MAJ*[<, +].*

This part of the thesis looks at regular quantifiers in the presence of a linear order and addition relation. In the presence of a linear order, addition and multiplication, most of these results are unknown, and are fundamental to understanding circuit complexity classes. A question, which can arise in the minds of the readers, might be what do we know when there is only the addition relation, or only linear order and multiplication or only the multiplication relation. We quickly go through these questions. Schweikardt's paper [Sch05] and her thesis [Sch01] give a detailed study of this area.

Theorem 7.1.5. *FO[<]⊊ FO[<, +]= FO[+].*
[GS66] FO[<, +] ⊊ FO[<, +, ×].
[Rob49, Lee03, Imm99, Sch05] FO[+, ×] = FO[<, ×] = FO[<, +, ×].
[Sch05] FO[×] cannot express <.

We now look at regular quantifiers and their connections to Circuit complexity classes. Later we look at the Crane Beach conjecture and how this concept can be used for proving lower bounds.

7.1.1 *Descriptive Complexity of Circuit classes*

The circuit family class AC^0 is defined as the family of languages recognized by constant depth polynomial sized family of *circuits* having unbounded fan-in *AND*, and *OR* gates. Similarly $ACC^0(p)$ is the family of languages recognized by constant depth polynomial sized family of circuits containing unbounded fan-in *AND*, *OR* and MOD_p for $p > 0$. Similarly $CC^0(p)$ corresponds to constant depth, polynomial size circuits with only MOD_p gates. $ACC^0(CC^0)$ is defined as the set of languages recognized by an $ACC^0(p)$ ($CC^0(p)$) family of circuits for some $p > 0$. The circuit class TC^0 corresponds to circuits with constant depth, polynomial size and having in addition to *AND* and *OR* gates *MAJ* (majority) gate. On the other hand NC^1 circuits are defined polynomial sized, log depth circuits containing *AND* and *OR* gates. There is an alternate characterization for NC^1. It is the family of languages recognized by constant depth, polynomial sized family of circuits which uses *AND*, *OR* and finite group gates. The reader can refer to the books [Vol99], [Juk12] to know more about these classes.

Results by Razborov [Raz89] and Smolensky [Smo87] shows that:

Theorem 7.1.6. *[Raz89, Smo87] If p is a prime number and q is a prime other than p then the language $\mathcal{L}_q$ is not contained in* $\mathrm{ACC}^0(p)$.

Hence we can infer the following: AC^0 is separated from $\mathrm{ACC}^0(p)$ for a $p > 0$ [FSS84]; there are languages in $\mathrm{CC}^0(p)$ which are not in AC^0; the classes $\mathrm{ACC}^0(p)$ and $\mathrm{ACC}^0(q)$ are different from each other if p and q are distinct primes. But relationships between most other classes are open. For example, we do not know whether CC^0 is different from ACC^0. In fact we do not know whether $\mathrm{CC}^0(6)$ contains AC^0 or whether $\mathrm{CC}^0(6)$ is even distinct from NP. These are among the biggest unsolved problems in circuit complexity.

Each of the above circuit classes have a model-theoretic characterization. It is known, from the results of Immerman [Imm87b, Imm87a], that the set of languages accepted by *non-uniform*-AC^0 circuits are exactly those definable by first order logic which uses an order and some *arbitrary* relations. We denote this logic by $\mathrm{FO}[<, \mathrm{Arb}]$, where Arb is the class of all relations possible on $\mathbb{N}$. On the other hand *dlogtime-uniform*-AC^0 circuits are exactly those definable (see Barrington et.al[BIS90]) by first order logic which uses order, addition and multiplication relations (denoted by $\mathrm{FO}[<, +, \times]$). First order logic with different built-in predicates can be seen as the complexity class AC^0 with different uniformity conditions. From here onwards we consider only *dlogtime-uniform* circuits and hence any circuit family we mention will be *dlogtime-uniform* unless otherwise stated. Behle and Lange [BL06] gives a notion of interpreting $\mathrm{FO}[<, +]$ as highly uniform circuit classes. Other circuit families also have model theoretic characterization. We have that the circuit family CC^0corresponds to $\textsc{mod}[<, +, \times]$, ACC^0 corresponds to $\mathrm{FO}\textsc{mod}[<, +, \times]$, TC^0 corresponds to $\textsc{maj}[<, +, \times]$, and NC^1 corresponds to $\textsc{group}[<, +, \times]$. The above characterization of the circuit classes come under Descriptive complexity (it studies how different complexity classes can be captured by different logics) of circuit classes. The books by Immerman [Imm99], Vollmer [Vol99] and Straubing [Str94] show the close connection between logics with monoid quantifiers and circuit classes. Table 6 identifies the language/complexity classes for logics with different quantifiers and relations.

		Relations		
		$[<]$	$[<, +]$	$[<, +, \times]$
Quantifiers	$\exists$	Aperiodic	Our Study	AC^0
	MOD_p	p-Solvable groups		$\mathrm{CC}^0(p)$
	MOD	Solvable Groups		CC^0
	$\exists, MOD_p$	p-Solvable Monoids		$\mathrm{ACC}^0(p)$
	$\exists, MOD$	Solvable Monoids		ACC^0
	S_5	Symmetric Group, S_5		NC^1
	$Group$	Groups		NC^1
	$\exists, Group$	Monoids		NC^1
		Algebraic characterization		**Circuit Complexity**

Table 6: Generalized quantifiers and Expressiveness

This helps us to look at the questions regarding separation of circuit classes from the descriptive complexity perspective. But no separation result has been made after the announcement of Smolensky's result. Razborov and Rudich [RR97] has analyzed the reason why these questions are hard. They show that no "*natural proof*" can prove the separation between these classes. Hence, as a first step, one can ask the question of separating the logics when the multiplication relation is not available. That is, can one separate $\text{MOD}[<,+]$ from $\text{FO}\text{MOD}[<,+]$? Is $\text{GROUP}[<,+]$ different from $\text{FO}\text{MOD}[<,+]$? Table 6 shows the algebraic characterization for each of the logic classes if only the linear order is present. Algebraic techniques can be used to show that these classes are separated from each other [Str94]. Hence the most natural question would be to understand the classes of languages accepted by the various logics when addition is also present.

In chapter 8, we give a powerful technique to prove lower bound results for $\text{FO}[<,+]$ extended with regular quantifiers. In fact we show that most of these classes are separated. The separation corresponds to algebraic properties of the quantifiers. The corollary 8.2.13 puts this in perspective.

7.1.2 *The Crane Beach conjecture*

As we have seen, there are just a handful of techniques available for proving lower bound results for circuit families. Most of these techniques are combinatorial in nature. Secondly, there has been very little understanding of the power the built-in predicates give to the logic classes. For example, how do we show that a certain language in *non-uniform*-$\text{AC}^0$is infact not in *dlogtime-uniform*-AC^0. In order to understand the expressive power of different relations, Thérien proposed (see Barrington et.al[BIL+05]), what came to be called the **Crane Beach conjecture**. In order to state the conjecture, we need to define a *neutral letter language*.

Definition 7.1.7. *Let $L \subseteq \Sigma^*$ be a language over the alphabet Σ. We say that a letter $\lambda \in \Sigma$ is a* neutral letter *for the language L if*

$$u\lambda v \in L \Leftrightarrow uv \in L$$

That is we can insert or delete the letter λ from any word, $w \in \Sigma^*$ without affecting its membership in L.

Let us look at an example.

Example 7.1.8. *(Parity) The following language $\mathcal{L}_2$ is a language with a neutral letter, where b is a neutral letter.*

$$\mathcal{L}_2 = \{w \in \{a,b\}^* \mid |w|_a \equiv (0 \ mod \ 2)\}$$

Here is another example:

Example 7.1.9. *(word problem over group) Look at a word problem over a group G. Then clearly the identity element of the group, 1_G is a neutral letter for the language.*

The *Crane Beach conjecture* states that

Conjecture 7.1.10. *A language with a neutral letter is definable in FO[<, Arb] iff it is definable in FO[<].*

The conjecture says that first order logic with arbitrary numerical predicates will collapse to first order logic with only linear ordering in the presence of a neutral letter. The idea is that, in the presence of a neutral letter, formulas cannot rely on the precise location of input letters and hence numerical predicates will be of little use. Nevertheless the conjecture was refuted by Barrington et. al [BIL+05]. In fact they show, using the fact (see Ajtai and Ben-Or [ABO84]) that *dlogtime-uniform*-AC^0 can count the number of occurrences of the letter a upto log of the input size, that the conjecture does not hold for the logic $FO[<,+,\times]$, i.e. first order logic with a linear order, addition, and multiplication relations. The search was then to find out for what logics and relations does the conjecture hold. So, in the most general form, the *Crane Beach conjecture* can be stated as follows. Let **NLL** denote the class of languages with neutral letters. Let $\mathcal{R}$ be a set of relations on $\mathbb{N}$. Let $\mathcal{S}$ be a subset of monoids. Then the Crane Beach conjecture says that [1]

Conjecture 7.1.11.

$$\mathcal{L}_{\mathcal{S}}[<,\mathcal{R}] \cap \mathbf{NLL} = \mathcal{L}_{\mathcal{S}}[<] \cap \mathbf{NLL}$$

In the same paper [BIL+05], the authors identify various logics where the *CBC* (short for Crane Beach conjecture) hold and various other logics where the *CBC* does not hold. For example. The Boolean closure of the Σ_1-fragment of FO[Arb] does satisfy the conjecture. That is $\mathcal{B}(\Sigma_1)[\mathrm{Arb}] \cap \mathbf{NLL} = \mathcal{B}(\Sigma_1)[<] \cap \mathbf{NLL}$. Lautemann, Tesson and Thérien [LTT06] considered modulo counting quantifiers. They show that $\mathcal{B}(\Sigma_1^{0,p})[\mathrm{Arb}] \cap \mathbf{NLL} = \mathcal{B}(\Sigma_1^{0,p})[<] \cap \mathbf{NLL}$. This is equivalent to showing that

Theorem 7.1.12. *[LTT06] Let p be a prime number. Then*

$$\mathrm{MOD}_p[\mathrm{Arb}] \cap \mathbf{NLL} = \mathrm{MOD}_p[<] \cap \mathbf{NLL}$$

Benedikt and Libkin [BL00b], in the context of collapse results in database theory, had shown that first order logic with only the addition and order relation satisfies the Crane Beach conjecture. A different proof of the result can be found in [BIL+05]. We show that this result can be generalized to any monoid quantifier. Let $\mathcal{S}$ be a subset of monoids. Our main result (Theorem 8.2.1) shows that the Crane Beach conjecture hold for the logic $\mathcal{L}_{\mathcal{S}}[<,+]$. That is:

$$\mathcal{L}_{\mathcal{S}}[<,+] \cap \mathbf{NLL} = \mathcal{L}_{\mathcal{S}}[<] \cap \mathbf{NLL}.$$

[1] $\mathcal{L}_{\mathcal{S}}$ be the logic closed under quantification, where the quantifiers are Lindström quantifiers are over some monoid in $\mathcal{S}$.

If S is an aperiodic monoid, then the Theorem is equivalent to the result of Benedikt and Libkin. Roy and Straubing [RS07] (used ideas of Benedikt and Libkin to) show that $FO\text{MOD}[<,+]$ in the presence of neutral letters collapse to $FO\text{MOD}[<]$. In the same paper they posed the question

Conjecture 7.1.13. *(posed in [RS07])*

$$\text{MOD}[<,+] \cap \mathbf{NLL} = \text{MOD}[<] \cap \mathbf{NLL}$$

This is proved by a corollary of our Theorem 8.2.1.

Our main Theorem can also be viewed from the Circuit complexity perspective. Our results therefore can be summarized as: every FO[<, +] uniform constant depth polynomial size circuit with gates that compute a product in $\mathcal{S}$ and that recognize a language with a neutral letter can be made FO[<]-uniform.

As a consequence of our Theorem 8.2.1 we are able to separate these highly uniform versions of circuit classes. For example: The theorem states that $\text{MOD}[<,+]$ definable languages with a neutral letter are also definable in $\text{MOD}[<]$. Since $\text{MOD}[<]$ cannot simulate existential quantifiers [Str94] we have that FO[<, +] and $\text{MOD}[<,+]$ are incomparable. In fact we show that no group quantifier can simulate existential quantifier if only addition is available.

Another corollary gives an alternate proof of the known result [RS07] that $\text{FOMOD}(m)[<,+]$ cannot count modulo a prime p, which does not divide m.

Another corollary shows that the *majority quantifier* cannot be simulated by group quantifiers if multiplication is not available, thus separating $\text{MAJ}[<,+]$ from $\text{FOGRP}[<,+]$. Barrington's theorem [Bar89] says that word problems over any finite group can be defined by the logic which uses only the S_5 group quantifier (the group whose elements are the set of all permutations over 5 elements) if addition and multiplication predicates are available. Our result show multiplication is necessary for Barrington's theorem to hold. In other words S_5 cannot define word problems over S_6 if only addition is available.

The interesting thing to note is how the *neutral* letter concept has turned out to useful for proving lower bound results for "highly" uniform circuit classes. The neutral letter has also been used in the past for showing non-expressibility results. It had been used for showing super linear lower bounds for bounded-width branching programs [BS91], super linear wires in circuit classes [KPT05] and in communication complexity [CKK+07]. The neutral letter concept is also closely related to *collapse-results* in database theory [BL00a].

7.1.3 *Presburger arithmetic extended with modulo counting*

Now look at *Presburger arithmetic*. Presburger arithmetic is first order logic over the arithmetic model $(\mathbb{N}, <, +, \equiv, 0, 1)$. An important characteristic of Presburger arithmetic

is that it allows *quantifier elimination*. That is for every formula in this logic, there exists a quantifier free formula which is equivalent to it. That is

Theorem 7.1.14. *[Pre29] Let $\phi(x_1, \ldots, x_k)$ be a formula in Presburger arithmetic. Then there exists a quantifier free formula* [2] *$\psi(x_1, \ldots, x_k)$ such that for every $n_1, \ldots, n_k \in \mathbb{N}$*

$$\phi(n_1, \ldots, n_k) \text{ is true } \Leftrightarrow \psi(n_1, \ldots, n_k) \text{ is true}$$

The above theorem also gives decidability of Presburger arithmetic. Presburger arithmetic satisfies other properties too. From the results of Ruhl [Ruh99a], Schweikardt [Sch05] we know that $FO = FO\text{UNC}$ over $(\mathbb{N}, <, +, \equiv, 0, 1)$.

Theorem 7.1.15. *[GS66, Ruh99a, Sch05] Presburger arithmetic is closed under unary counting (and hence modulo counting) quantifiers.*

7.2 SATISFIABILITY

Let us look at the satisfiability questions. By an old result of Robinson [Rob58], we know that the satisfiability of first order logic with the addition relation, FO[<, +] over an alphabet of size greater than or equal to 2, is undecidable.

Theorem 7.2.1. *[Rob58] Satisfiability of FO[<, +] is undecidable over a two letter alphabet.*

A more recent proof can be found in [Lan04b]. We strengthen this claim to show that even the two variable fragment of this logic is undecidable. In particular we look at the logic $\text{FO}^2[<, succ, y = 2x, 1]$ over words and show that it is undecidable.

From [LMSV01] we know that FOUNC[<] can define addition. From the above discussion, we have that FOUNC[<] is undecidable.

Theorem 7.2.2. *[Rob58] Satisfiability of FOUNC[<] is undecidable.*

A closely related logic is $\text{FO}^2\text{UNC}[<, succ]$ over words. We show that the satisfiability problem is undecidable for this logic.

One can wonder, whether FO[<, +] is decidable when the alphabet size is 1. Observe that this logic corresponds to Presburger arithmetic.

Presburger's classic quantifier elimination result [Pre29] showed that FO over $(\mathbb{N}, <, +)$ is decidable. See Theorem 7.1.14. A second proof of this result, due to Büchi [Bö60], goes via interpretation on finite words, giving nonelementary decidability (see Bruyère, Hansel, Michaux and Villemaire [BHMV94]).

[2] There is infact an effective translation of a formula into its equivalent quantifier free formula.

Fischer and Rabin gave a superexponential lower bound for Presburger arithmetic [FR74]. Analysis of Presburger's proof by Cooper [Coo72], Oppen [Opp78], Ferrante and Rackoff [FR79] and Berman [Ber80] showed that the complexity of satisfiability is $\text{ATIME}[2^{2^{O(n)}}, O(n)]$, which is an alternating machine using a linear number of alternations and taking double exponential time. A matching lowerbound was also given by Berman [Ber80].

Theorem 7.2.3. *[FR79, Ber80] Satisfiability of Presburger arithmetic is* $\text{ATIME}[2^{2^{O(n)}}, O(n)]$*-complete.*

We look at Presburger arithmetic extended with modulo counting quantifiers. We give a *Ferrante-Rackoff* analysis and an elementary 2EXPSPACE upper bound for the logic FOMOD over $(\mathbb{N}, <, +)$, which is slightly above Berman's lower bound.

In Chapter 8 we look at the expressiveness of first order logic with addition relation, extended with regular (monoid) quantifiers.

In Chapter 9 we look at the satisfiability of various fragments of first order logic with addition, extended with modulo counting quantifiers.

8

FOGRP[<, +] EXPRESSIVENESS (LOWER BOUNDS RESULTS)

8.1 INTRODUCTION

In this chapter we will look at the expressiveness questions for FO[<, +] extended with various group quantifiers.

We present our main theorem and its corollaries in Section 8.2 followed by a section 8.3 with the Proof Strategy and then section 8.4, which gives the proof of Theorem 8.4.8. Section 8.5 which is our main contribution shows how to replace group quantifiers by its active domain version.

8.2 RESULTS

Recall that **NLL** is the set of all neutral letter language, where the neutral letter is λ. Let $\mathcal{S} \subseteq \mathcal{M}$ be any set of monoids. We show that the Crane Beach conjecture is true for the logic $\mathcal{L}_{\mathcal{S}}[<, +]$.

Theorem 8.2.1 (Main Theorem)**.** *Let $\mathcal{S} \subseteq \mathcal{M}$. Then*

$$\mathcal{L}_{\mathcal{S}}[<, +] \cap \mathbf{NLL} = \mathcal{L}_{\mathcal{S}}[<] \cap \mathbf{NLL}$$

The proof of this theorem is given in Section 8.4.

8.2.1 *Non definability Results*

Theorem 8.2.1 give us the following corollaries.

Corollary 8.2.2. *All languages with a neutral letter in $\mathcal{L}_{\mathcal{M}}[<, +]$ are regular.*

Proof. By Theorem 8.2.1 we know that all languages with a neutral letter in $\mathcal{L}_{\mathcal{M}}[<, +]$ can be defined in $\mathcal{L}_{\mathcal{M}}[<]$ which by Lemma 2.5.3 is the set of all regular languages. □

Recall that a monoid M *divides* a monoid N if M is a morphic image of a submonoid of N.

Corollary 8.2.3. *Let $\mathcal{S} \subseteq \mathcal{G}$. Let G be a simple group that does not divide any monoid M in $\mathcal{S}$. Then the word problem over G is not definable in $\mathcal{L}_{\mathcal{S}}[<,+]$.*

Proof. The word problem over G has a neutral letter. The result now follows from Theorem 8.2.1 and Lemma 2.5.3. □

It is known that the majority quantifier can be simulated by the non-solvable group S_5 if both multiplication and addition are available [Vol99]. We show that multiplication is necessary to simulate majority quantifiers.

Corollary 8.2.4. $\text{MAJ}[<] \not\subseteq \mathcal{L}_{\mathcal{M}}[<,+]$.

Proof. Consider the language $L \subseteq \{a,b,c\}^*$ consisting of all words with an equal number of a's and b's. L can be proven to be definable in $\text{MAJ}[<]$. Also note that c is a neutral element for L. By Corollary 8.2.2, and the fact that L is nonregular, we know that L is not definable in $\mathcal{L}_{\mathcal{M}}[<,+]$. □

Barrington's theorem [Bar89] says that the word problem of any finite group can be defined in the logic $\mathcal{L}_{S_5}[<,+,*]$. The following theorem shows that multiplication is necessary for Barrington's theorem to hold.

Corollary 8.2.5. *The word problem over the group S_6 is not definable in $\mathcal{L}_{S_5}[<,+]$. In fact there does not exist any one finite monoid M such that all regular languages can be defined in $\mathcal{L}_M[<,+]$.*

Proof. A_6 is a simple subgroup of S_6, which does not divide S_5. From Corollary 8.2.3 it follows that the word problem over S_6 is not definable in $\mathcal{L}_{S_5}[<,+]$.
For any finite monoid M, there exists a simple group G such that G does not divide M and hence the word problem over G is not definable in $\mathcal{L}_M[<,+]$. □

Let L_p be the set of all words $w \in \{0,1\}^*$ such that the number of occurrences of 1 in w is equal to $0 \pmod p$. Then we get the result in [RS07] that L_p is not definable in $FO\text{MOD}_m[<,+]$, if p is a prime which does not divide m.

Corollary 8.2.6 ([RS07])**.** *If p is a prime which does not divide m, then L_p is not definable in $FO\text{MOD}_m[<,+]$.*

Proof. Let L_p be definable in $FO\text{MOD}_m[<,+]$. Since 0 is a neutral letter in L_p, Theorem 8.2.1 says L_p is also definable in $FO\text{MOD}_m[<]$. Due to Lemma 2.5.3 and [Str94], this is a contradiction. □

It is known that languages accepted by CC^0 circuits are exactly those which are definable by $\mathcal{L}_{\text{MOD}}[<,+,*]$ formulas [Vol99]. On the other hand, it is an open question whether the language 1^* can be accepted by the circuit complexity class CC^0 [Str94].

To progress in this direction Roy and Straubing [RS07] had posed the question of whether $1^* \notin \mathcal{L}_{\text{MOD}}[<,+]$. Below we show that this is the case.

Corollary 8.2.7. $1^* \notin \mathcal{L}_{\text{MOD}}[<,+]$. *In fact* $1^* \notin \mathcal{L}_{\mathcal{G}}[<,+]$.

Proof. The minimal monoid which can accept 1^* is U_1 and clearly 1 is a neutral letter. By Theorem 8.2.1 if there is a formula in $\mathcal{L}_{\mathcal{G}}[<,+]$ which can define 1^*, then $\mathcal{L}_{\mathcal{G}}[<]$ can also define 1^*. From Lemma 2.5.3 it follows that the monoid U_1 divides a group. But this is a contradiction [Str94]. □

Behle and Lange [BL06] give a notion of interpreting $\mathcal{L}_{\mathcal{S}}[<,+]$ as highly uniform circuit classes. As a consequence we can interpret the following results as a separation of the corresponding circuit classes.

Corollary 8.2.8. *The following separation results hold, for all* $m > 1$

- $FO[<,+] \not\subseteq \text{MOD}[<,+]$.
- $\text{MOD}_m[<,+] \not\subseteq FO[<,+]$.
- $FO[<,+] \subsetneq FO\text{MOD}_m[<,+] \subsetneq FO\text{MOD}[<,+]$
- $FO\text{MOD}[<,+] \subsetneq FO\text{GROUP}[<,+]$
- $\text{MAJ}[<,+] \not\subseteq FO\text{GROUP}[<,+]$

Proof. We will show that $FO[<,+] \not\subseteq \text{MOD}[<,+]$. Assume not. Consider the language $L = 1* \subseteq \{0,1\}^*$. It is clearly a neutral letter language (1 is the neutral letter). Also this language is expressible in FO[<]. We get a contradiction since by Corollary 8.2.7 we know 1^* is not in $\text{MOD}[<,+]$. □

8.2.2 *Decidability of Regular languages in* $L_{\mathcal{S}}[<,+]$

We now look at regular languages definable by the logic $L_{\mathcal{S}}[<,+]$, for an $\mathcal{S} \subseteq \mathcal{M}$. We first show that this logic is closed under quotienting and under inverse length preserving morphims. We do not give the proofs of these claims, since they follow the standard technique. One can refer to [RS07] for the proof.

Lemma 8.2.9. *Let* $S \subseteq \mathcal{M}$ *and* Σ *be a finite alphabet. Let* $L \subseteq \Sigma^*$ *be definable in* $L_{\mathcal{S}}[<,+]$ *and* $u, v \in \Sigma^*$. *Then* $u^{-1}Lv^{-1}$ *is also definable in* $L_{\mathcal{S}}[<,+]$.

Lemma 8.2.10. *Let $S \subseteq \mathcal{M}$. Let Σ, Γ be finite alphabets and let $h : \Gamma^* \to \Sigma^*$ be a homomorphism such that $h(\Gamma) \subseteq \Sigma^r$ for some fixed $r > 0$. If $L \subseteq \Sigma^*$ is definable in $L_S[<,+]$, then $h^{-1}(L) \subseteq \Gamma^*$ is also definable in $L_S[<,+]$.*

We now give an algebraic characterization for regular languages definable by $\mathcal{L}_S[<,+]$. Recall from the Preliminaries chapter that $\mathcal{L}_S[\mathrm{REG}]$ is defined as $\mathcal{L}_S[<, succ, \equiv]$ where $\equiv$ are modulo predicates. [1]

Theorem 8.2.11. *Let $\mathcal{S} \subseteq \mathcal{M}$ be a set of monoids. Let $L \subseteq \Sigma^*$ be a regular language, which is accepted by a morphism $h : \Sigma^* \to \mathcal{V}$, where $\mathcal{V}$ is a semigroup. Then the following are equivalent.*

1. *L is definable in $\mathcal{L}_S[<,+]$*
2. *For all $k \in \mathbb{N}$, every group in $h(\Sigma^k)$ divides a monoid in $bpc(\mathcal{S})$.*
3. *L is definable in $\mathcal{L}_S[\mathrm{REG}]$*

Proof. $(1 \Rightarrow 2)$: Consider a $k \in \mathbb{N}$ and a $G \in h(\Sigma^k)$. We first look at the language $h^{-1}(1_G)$. It is well known (for example, [Str94]) that $h^{-1}(1_G)$ can be written as a finite boolean combination of languages of the form $u^{-1}Lv^{-1}$, for strings $u, v \in \Sigma^*$. Now consider a new alphabet

$$\Gamma = \{a_w \mid w \in \Sigma^k, h(w) \in G\}$$

We can now define a mapping $f : \Gamma \to \Sigma^k$ as $f(a_w) = w$. Consider the language $L' = f^{-1}(h^{-1}(1_G))$. From Lemma 8.2.9 and Lemma 8.2.10 we know that L' is definable in $L_S[<,+]$. But note that L' is a language with a neutral letter. The letter a_w, where $h(w) = 1_G$ acts as a neutral letter. Therefore L' is a language definable in $L_S[<]$, which from Krohn Rhodes theorem 2.3.4 implies that G divides a monoid in $bpc(\mathcal{S})$.
$(2 \Rightarrow 3)$: Let us list down all the sets, $h(\Sigma), h(\Sigma^2), \ldots$. Since the subsets of $\mathcal{V}$ are finite, there exists $q, r \in \mathbb{N}$ such that $h(\Sigma^q) = h(\Sigma^{q+r})$. Choosing an $l = \mathrm{lcm}\{q, r\}$ we get that $h(\Sigma^l) = h(\Sigma^{2l})$. We can now split L into the following parts.

$$L = \bigcup_{u \in \Sigma^{<l}} u.\left(u^{-1}L \cap (\Sigma^l)^*\right)$$

Let us denote by $L_u = u^{-1}L \cap (\Sigma^l)^*$. First let us show that, if we can write L_u in $\mathcal{L}_S[\mathrm{REG}]$, then we can write $u.L_u$ also in $\mathcal{L}_S[\mathrm{REG}]$. Let us assume that the sentence $\phi_u \in \mathcal{L}_S[\mathrm{REG}]$ models the language L_u. Let $u = a_1 \ldots a_{l'}$, where $l' < l$. Then the following formula models the language $u.L_u$.

$$\phi_u[> l'] \wedge a_1(1) \wedge \cdots \wedge a_{l'}(l')$$

Above $\phi_u[> l']$ is a formula got by relativizing all variables in ϕ_u by $> l'$. Now since L is a disjunction of languages of the form uL_u we have that L can be written in $\mathcal{L}_S[\mathrm{REG}]$.

[1] Theorem 8.2.11 was mentioned for solvable monoids. One of the referees pointed out that the proof goes through for all finite monoids.

We need to show that L_u is also definable. Consider the following alphabet $\Gamma = \{a_w \mid w \in \Sigma^l\}$. Let $M = h(\Sigma^l)$. Since $M = h(\Sigma^{2l})$ we have that M is closed under concatenation and therefore a monoid. Therefore we have that word problems over M are definable in $\mathcal{L}_M[<]$. Finally using Lemma 8.2.10 we get that L_u is definable in $\mathcal{L}_M[\mathrm{REG}]$.
$(3 \Rightarrow 1)$: This holds, since addition can simulate all the predicates in REG. □

Then we can identify the set of regular languages definable in $\mathcal{L}_{\mathcal{S}}[<,+]$ when $\mathcal{S}$ is a set of monoids.

Corollary 8.2.12. *Let $\mathcal{S}$ be a set of monoids. Then*

$$\mathcal{L}_{\mathcal{S}}[<,+] \cap \textit{ Regular languages } = \mathcal{L}_{\mathcal{S}}[\mathrm{REG}]$$

Let $\mathcal{S}$ be a set of monoids such that, given a group G, it is decidable if G divides a monoid in $bpc(\mathcal{S})$. Then, given a regular language L, it is decidable if $L \in \mathcal{L}_{\mathcal{S}}[<]$. Then our main theorem gives us that it is decidable if $L \in \mathcal{L}_{\mathcal{S}}[<,+]$.

Corollary 8.2.13. *Let $\mathcal{S}$ be a set of monoids such that, given a monoid G, it is decidable if G divides a monoid in $bpc(\mathcal{S})$. Then, given a regular language L, it is decidable if $L \in L_{\mathcal{S}}[<,+]$.*

Proof. Since L is a regular language definable in $L_{\mathcal{S}}[<,+]$. Then L has the alternate characterization given by Theorem 8.2.11. Let h be a morphism which accepts L. Then we know that there exists $t, r \in \mathbb{N}$, such that $h(\Sigma^t) = h(\Sigma^{t+r})$, which implies that we can list down all the sets in $h(\Sigma^k)$, for $k \leq t + r$ and the groups in these sets. The claim now follows from the fact that there exists an algorithm to check whether each of these groups divide a monoid in $bpc(\mathcal{S})$. □

The above theorem for $FO\mathrm{MOD}[<,+]$ was proved in [RS07] and the question when $\mathcal{S} = MOD$ was left open. The following corollary answers this special case.

Corollary 8.2.14. *Given a regular language L, the question whether L is definable in* $\mathrm{MOD}[<,+]$ *is decidable.*

8.3 PROOF STRATEGY

For the purpose of proof we work over infinite strings which contain finite number of non-neutral letters. Our general proof strategy is similar to Benedikt and Libkin [BL00b] or Roy and Straubing [RS07] and consists of three main steps.

1. Given a formula $\phi \in \mathcal{L}_{\mathcal{S}}[<,+]$, we show that ϕ is "*weakly equivalent*" to an "active domain formula" $\phi' \in \mathcal{L}_{\mathcal{S}}[<,+]$. Active domain formulas quantify only over *non-neutral letter positions*. Our major contribution (Theorem 8.4.8) is in showing this step.

2. Any active domain formula $\phi' \in \mathcal{L}_S[<,+]$ is weakly equivalent to an active domain formula $\phi'' \in \mathcal{L}_S[<]$. This step follows from an application of Ramsey theory (Theorem 8.4.9).

3. All active domain formulas in $\mathcal{L}_S[<]$ accept languages with a neutral letter. This is an easy observation given by Lemma 8.4.10.

Finally using these three steps we can show that if a formula in $\mathcal{L}_S[<,+]$ is weakly equivalent to an active domain formula in $\mathcal{L}_S[<]$, then it is infact equivalent. This proves our main Theorem.

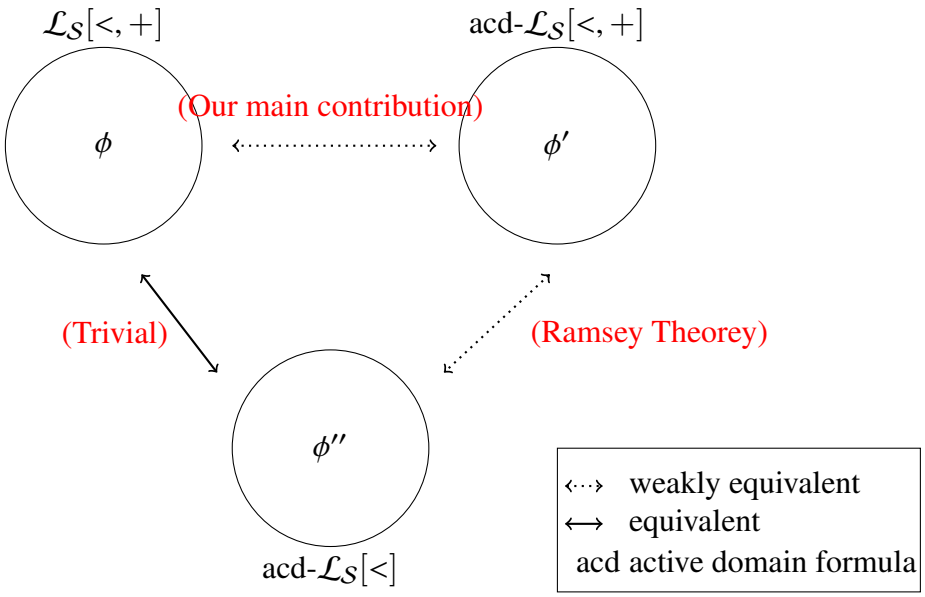

Figure 9: The three step Proof strategy.

The main step is to build an active domain formula, that is step 1. Here we show how to simulate a general quantifier by an active domain formula. In the case of FO$[<,+]$, the existential quantifier is equivalent to the monoid quantifier U_1. The monoid U_1 is a commutative and idempotent monoid. Hence neither the order in which the quantifier runs over the positions of the word is important, nor does it matter if positions are queried multiple times. In [RS07] this idea was extended in such a way that in the simulation of the MOD quantifier (again a commutative monoid), every position is taken into account exactly once. In their construction while replacing a MOD quantifier they need to add additional FO quantifiers and hence their construction only allows to replace a MOD$[<,+]$ formula by an active domain FOMOD$[<,+]$ formula. In this paper, we construct a formula that takes every position into account exactly once and in the correct order. Moreover we do not introduce any new quantifier, but use only the quantifier that is replaced. This enables us to show the Crane Beach conjecture for logics whose quantifiers have a non-commutative monoid or are groups. For example MOD$[<,+]$, GROUP$[<,+]$, and FOGROUP$[<,+]$.

In contrast to previous work, we do not construct an equivalent active domain formula, but only a formula that is equivalent for certain domains. We show that it is in general sufficient to show this for one infinite domain.

8.4 PROOF OF THE MAIN THEOREM

In this section we handle the general proof steps as in [BL00b] and [RS07] of removing the plus predicate from the formula in the presence of a neutral letter. We show that all these results go through even in the presence of general Lindström quantifiers. The new crucial step is Lemma 8.4.6 where we convert a group quantifier to an active domain formula without introducing any other quantifiers. The proof of this lemma is deferred to the next section.

8.4.1 *Definitions*

Let $\mathcal{S} \subseteq \mathcal{M}$ be any nonempty set. To prove Theorem 8.2.1 we will consider the more general logic, $\mathcal{L}_{\mathcal{S}}[<, +, \{\equiv_q : q > 1\}]$ over the alphabet Σ. In this logic $+$ is a binary function, and $a \equiv_q b$ means q divides $b - a$. We will denote this logic by $\mathcal{L}_{\mathcal{S}}[<, +, \equiv]$. The relations $<$ and $\equiv_q$ are both definable using $+$. All languages recognized by this logic are definable in $\mathcal{L}_{\mathcal{S}}[<, +]$. The reason for introducing these new relations is to use a quantifier elimination procedure.

For the purpose of the proof we assume that the neutral letter language defined by a formula $\phi \in \mathcal{L}_{\mathcal{S}}[<, +]$ is a subset of $\Sigma^*\lambda^\omega$, where λ is the neutral letter. The idea is to work with infinite words, where the arguments are easier, since the variable range is not bounded by the word length. The results, still hold for finite words.

Definition 8.4.1. *The non-neutral letter positions of a word w, denoted by $nnp(w)$ is the set of all positions where the letter λ does not appear.*

$$nnp(w) = \{i \mid w(i) \neq \lambda\}$$

Observe that $nnp(w)$ is also defined with respect to a letter, namely λ.

Let us look at the following example.

Example 8.4.2. *Let $w = a\lambda\lambda\lambda ba\lambda b\lambda\lambda a$. Then $nnp(w) = \{1, 5, 6, 8, 11\}$.*

Definition 8.4.3. *Let $X \subseteq \mathbb{N}$ be an infinite set. We say that a formula ϕ' is X-weakly equivalent to a formula $\phi(x_1, \ldots, x_t)$ if there exists an infinite set $Y \subseteq X$, such that for all words $w \in \Sigma^*\lambda^\omega$, with $nnp(w) \subseteq Y$ and for all $a_1, \ldots, a_t \in \mathbb{N}$, we have that*

$$w \models \phi(a_1, \ldots, a_t) \Leftrightarrow w \models \phi'(a_1, \ldots, a_t)$$

In the above definition we say that Y collapse *ϕ to ϕ'.*

We say that a formula ϕ' is weakly equivalent *to a formula $\phi(x_1, \ldots, x_t)$ if for all infinite subsets $X \subseteq \mathbb{N}$, there exists an infinite set $Y \subseteq X$, such that for all words $w \in \Sigma^* \lambda^\omega$, with $nnp(w) \subseteq Y$ and for all $a_1, \ldots, a_t \in \mathbb{N}$, we have that*

$$w \models \phi(a_1, \ldots, a_t) \Leftrightarrow w \models \phi'(a_1, \ldots, a_t)$$

In this latter case we say that ϕ collapses *to ϕ'.*

It is clear from the definition that if ϕ' is weakly equivalent to ϕ, then ϕ' is X-weakly equivalent to ϕ for any infinite set X. The other direction, namely if ϕ' is X-weakly equivalent to ϕ need not imply that ϕ' is weakly equivalent to ϕ.

We now define a special class of formulas, defined syntactically.

Definition 8.4.4. *Let an* active domain formula *over a letter $\lambda \in \Sigma$ be a formula where all quantifiers are of the form: $Q_M^m x\ \neg\lambda(x)\langle\phi_1, \ldots, \phi_K\rangle$. That is the variables are quantified over the "*active domain*", the positions which does not contain the letter λ.*

Recall from the Preliminaries that $Q_M^m x\ \psi\langle\phi_1, \ldots, \phi_K\rangle$ stands for the formula $Q_M^m x\ \langle\phi_1 \wedge \psi, \ldots, \phi_K \wedge \psi\rangle$. Note that an active domain formula is defined with respect to a particular letter. Here we use the letter $\lambda \in \Sigma$, since the active domain formulas we consider will always be with respect to the neutral letter of the language. Observe also that an active domain formula need not always define a language with a neutral letter.

Example 8.4.5. *Consider the following formula in FO[+].*

$$\exists x\ \neg\lambda(x)\ \wedge\ \exists y\ \neg\lambda(y)\ \wedge\ \exists z\ \neg\lambda(z)\ \ ((a(x) \wedge b(y) \wedge c(z)) \wedge x = y + z)$$

Clearly it is an active domain formula but does not define a language with a neutral letter.

We will later see that if the only relation is a linear ordering, then the active domain formulas define a logic with a neutral letter.

The positions in a word which the quantifiers in an active domain formula access are going to be positions which have a non-neutral letter in it.

8.4.2 *The Proof*

This subsection follows the proof outline as given in Figure 9.

We first show that any formula $\phi \in \mathcal{L}_S[<, +, \equiv]$ will be weakly equivalent to an active domain formula $\phi' \in \mathcal{L}_S[<, +, \equiv]$. The results by Benedikt and Libkin [BL00b], and Roy and Straubing [RS07] show that for all formulas $\phi \in \mathcal{L}_{\text{MOD} \cup \{U_1\}}[<, +, \equiv]$ there exists an active domain formula ϕ' in that logic, such that for all words $w \in \Sigma^* \lambda^\omega$, $w \vDash \phi \Leftrightarrow w \vDash \phi'$. They assume no restriction on the non-neutral positions of w. Observe that our collapse

result is different from theirs. We prove that if we consider only words, whose non-neutral positions come from a particular subset of $\mathbb{N}$, then $\phi \in \mathcal{L}_{\mathcal{S}}[<,+,\equiv]$ is equivalent to an active domain formula $\phi' \in \mathcal{L}_{\mathcal{S}}[<,+,\equiv]$. In other words, we are not concerned about the satisfiability of those words whose non-neutral positions are not from that particular subset.

Let us first consider formulas with an outermost group quantifier, $G \in \mathcal{S}$.

Lemma 8.4.6. *Let $\phi = Q^m_G z\langle\phi_1,\ldots,\phi_K\rangle$ be in $\mathcal{L}_{\mathcal{S}}[<,+,\equiv]$. Let us assume that there exists active domain formulas $\phi'_1,\ldots,\phi'_k$ in the same logic such that for all $i \leq K$ we have ϕ_i is weakly equivalent to ϕ'_i.*
Then ϕ is weakly equivalent to an active domain formula $\phi' \in \mathcal{L}_{\mathcal{S}}[<,+,\equiv]$.

The proof of Lemma 8.4.6 will be given in Section 8.5. Benedikt and Libkin [BL00b] give a similar theorem for the monoid U_1 (the existential quantifier).

Lemma 8.4.7 ([BL00b])**.** *Let $\phi = Q^1_{U_1} z\langle\phi_1\rangle$ be a formula in $\mathcal{L}_{\mathcal{S}}[<,+,\equiv]$. Let us assume that the formula ϕ_1 is weakly equivalent to an active domain formula ϕ'_1 in the same logic. Then ϕ is weakly equivalent to an active domain formula $\phi' \in \mathcal{L}_{\mathcal{S}}[<,+,\equiv]$.*

The following theorem proves the first step of our 3 step proof strategy of Figure 9.

Theorem 8.4.8. *Let $\phi \in \mathcal{L}_{\mathcal{S}}[<,+,\equiv]$. Then there exists an active domain formula $\phi' \in \mathcal{L}_{\mathcal{S}}[<,+,\equiv]$ such that ϕ is weakly equivalent to ϕ'.*

Proof. Let $\phi \in \mathcal{L}_{\mathcal{S}}[<,+,\equiv]$. We first claim that we can convert ϕ into a formula which uses only groups and U_1 as quantifiers. This follows from the Krohn-Rhodes decomposition theorem for monoids that every monoid can be decomposed into block products over groups and U_1. This decomposition can then be converted back into a formula using the groups and U_1 as quantifiers [Str94].

So without loss of generality we can assume ϕ has only group or U_1 quantifiers. Now we prove by induction on the quantifier depth. For the base case, let ϕ be a quantifier free formula. It is an active domain formula and therefore the claim holds. Let the claim be true for all formulas with quantifier depth $< d$. Lemma 8.4.6 and Lemma 8.4.7 show that the claim is true for formulas of type $\phi = Q^m_M z\langle\phi_1,\ldots,\phi_K\rangle$ with quantifier depth d, when M is a group or U_1 respectively. We are now left with proving that the claim is closed under conjunction and negation. So assume that formulas ϕ_1,ϕ_2 is weakly equivalent to ϕ'_1,ϕ'_2 respectively. That is, for all $X \subseteq \mathbb{N}$, there exist $\mathcal{R}_{\phi_1} \subseteq X, \mathcal{R}_{\phi_2} \subseteq \mathcal{R}_{\phi_1}$ such that $\mathcal{R}_{\phi_1}$ is weakly equivalent to ϕ_1 to ϕ'_1 and $\mathcal{R}_{\phi_2}$ collapses ϕ_2 to ϕ'_2. Then it is easy to see that $\mathcal{R}_{\phi_2}$ collapses $\phi_1 \wedge \phi_2$ to $\phi'_1 \wedge \phi'_2$ and $\mathcal{R}_{\phi_1}$ collapses $\neg\phi_1$ to $\neg\phi'_1$. □

We have shown above that all formulas in $\mathcal{L}_{\mathcal{S}}[<,+,\equiv]$ can be collapsed to active domain formulas. Now using a *Ramsey* type argument we obtain that addition is useless, giving us a formula in $\mathcal{L}_{\mathcal{S}}[<]$. This corresponds to the second step in our three step proof strategy.

Let R be any set of relations on $\mathbb{N}$ and let $\phi(x_1, \ldots, x_t)$ be an active domain formula in $\mathcal{L}_S[R]$. Let $X \subseteq \mathbb{N}$ be an infinite set. Then there exists a formula ϕ' in $\mathcal{L}_S[<]$ such that ϕ is X-weakly equivalent to ϕ'.

Weak equivalence for first order logic has been considered by Libkin [Lib04], where the notion of weak equivalence is known as *Ramsey property*. We show that this result can be extended to our logic.

Theorem 8.4.9. *Let R be a set of relations on $\mathbb{N}$. Let $X \subseteq \mathbb{N}$ be an infinite set. Then every active domain sentence in $\mathcal{L}_S[R]$ is X-weakly equivalent to an active domain formula in $\mathcal{L}_S[<]$.*

Proof. Let $\phi \in \mathcal{L}_S[R]$ be an active domain sentence. We now prove by induction on the structure of the formula. Let $P(x_1, \ldots, x_k)$ be a term in ϕ. Consider the infinite complete hypergraph, whose vertices are labeled by numbers from X and whose edges are all k tuple of vertices. Let $P(a_1, \ldots, a_k)$ be true, for $a_1, \ldots, a_k \in X$ and let the *order type* of $a_1, \ldots, a_k$ be o. Then we color the edge $(a_1, \ldots, a_k)$ by the quantifier free formula on $x_1, \ldots, x_k$ which describes the order type o. For example, if the order type is $a_2 = a_3 < a_1 < \cdots < a_k$, then the formula will be $x_2 = x_3 < x_1 < \cdots < x_k$. Observe that an edge can have multiple colors but the total number of different colorings possible is dependent only on k (a constant). Ramsey theory, now gives us that there exists an infinite set $Y \subseteq X$, such that the induced subgraph on the vertices in Y will have a monochromatic color, ie. all the edges will be colored using the same color or in other words, there exists an order type which is true for all the edges in this subgraph. Let us assume that the edges in Y are colored $x_1 < x_2 < \cdots < x_k$. Then for all $a_1, \ldots, a_t \in Y$

$$a_1, \ldots, a_t \models P(x_1, \ldots, x_k) \Leftrightarrow a_1, \ldots, a_t \models x_1 < x_2 < \cdots < x_k$$

This shows that $P(x_1, \ldots, x_k)$ satisfies the Ramsey property and thus all atomic formulas satisfy the Ramsey property. We now show that Ramsey property is preserved while taking Boolean combination of formulas. Consider the formula $\phi_1(x_1, \ldots, x_k) \wedge \phi_2(x_1, \ldots, x_k)$. We know that by induction hypothesis there exists a formula ψ_1 and an infinite set X such that for all $a_1, \ldots, a_k \in X$, $w \models \phi_1(a_1, \ldots, a_k) \Leftrightarrow w \models \psi(a_1, \ldots, a_k)$. We can now find an infinite set $Y \subseteq X$ and a formula ψ_2 such that the Ramsey property holds for the formula ϕ_2. Therefore for all $a_1, \ldots, a_k \in Y$

$$w, a_1, \ldots, a_k \vDash \phi_1 \wedge \phi_2 \Leftrightarrow w, a_1, \ldots, a_k \vDash \psi_1 \wedge \psi_2$$

Similarly we can show that the Ramsey property holds for disjunctions and negations. We need to now show that active domain quantification also preserves Ramsey property. So let X be an infinite subset of $\mathbb{N}$ and let

$$\phi(\vec{x}) = Q_M^m z \, \neg\lambda(z) \, \langle \phi_1(z, \vec{x}), \ldots, \phi_K(z, \vec{x}) \rangle$$

be a formula in $\mathcal{L}_S[R]$. By induction hypothesis we know that there exists an infinite set $Y_1 \subseteq X$ and an active domain formula $\psi_1 \in \mathcal{L}[<]$ such that for all $\vec{a} \in Y_1^t$ the Ramsey property is satisfied. That is $w \models \phi_1(\vec{a}) \Leftrightarrow w \models \psi_1(\vec{a})$. Now for ϕ_2, using the infinite set

Y_1 we can find an infinite set $Y_2 \subseteq Y_1$ and a formula ψ_2 satisfying the Ramsey property. Continuing like this will give us a set Y_K and formulas $\psi_1, \ldots, \psi_K$ such that $\forall j \leq K$ and for all $w \in \Sigma^* \lambda^\omega$ with $\mathrm{nnp}(w) \subseteq Y_K$, we have that $\forall b \in Y_K, \vec{a} \in Y_K^t$, $w \vDash \phi_j(b, \vec{a}) \Leftrightarrow w \vDash \psi_j(b, \vec{a})$. Hence we also have that $\forall j \leq K$

$$\{b \in Y_K \mid w \vDash \phi_j(b, \vec{a})\} = \{b \in Y_K \mid w \vDash \psi_j(b, \vec{a})\}$$

Therefore for the formula $\psi = Q_M^m z\, \neg\lambda(z)\, \langle \psi_1, \ldots, \psi_K \rangle$, we have $\forall w$ where $\mathrm{nnp}(w) \subseteq Y_K$ and $a_1, \ldots, a_t \in Y_K$ that

$$w \vDash \phi(a_1, \ldots, a_t) \Leftrightarrow w \vDash \psi(a_1, \ldots, a_t)$$

Observe that ψ is an active domain formula in $\mathcal{L}_S[<]$. □

We continue with the third step of our three step proof strategy.

Lemma 8.4.10. *Every active domain sentence in $\mathcal{L}_S[<]$ defines a language with a neutral letter.*

Proof. Let $\phi \in \mathcal{L}_S[<]$ be an active domain formula over letter $\lambda \in \Sigma$. That is, all quantifiers are relativized over $\neg\lambda(x)$. Let $w \in \Sigma^\omega$. Let $w' \in \Sigma^\omega$ got by inserting letter λ in w at arbitrary positions. Let $n_1 < n_2 < \ldots$ belong to $\mathrm{nnp}(w)$ and $m_1 < m_2 < \ldots$ be in $\mathrm{nnp}(w')$. Let $\rho : \mathrm{nnp}(w) \to \mathrm{nnp}(w')$ be the bijective map $\rho(n_i) = m_i$. We show that for any subformula ψ of ϕ and any $\vec{t} \in \mathrm{nnp}(w)^s$, we have that $w, \vec{t} \vDash \psi \Leftrightarrow w', \rho(\vec{t}) \vDash \psi$. Since the variables quantify only over the active domain, the claim holds for the atomic formula $x > y$, because $n_i > n_j$ iff $\rho(n_i) > \rho(n_j)$ for any i, j. Similarly the claim also hold for all other atomic formulas $x < y, x = y$ and $a(x)$ for an $a \in \Sigma$. The claim remains to hold under conjunctions, negations and active domain quantifications. Hence $w \models \phi \Leftrightarrow w' \models \phi$. This proves that λ is a neutral letter for $L(\phi)$. □

Now we can prove our main theorem. This step shows that if an active domain formula $\phi' \in \mathcal{L}_S[<]$ is X-weakly equivalent to a formula $\phi \in \mathcal{L}_S[<, +]$ then ϕ' is infact equivalent to ϕ.

Proof of Theorem 8.2.1. Let $\phi \in \mathcal{L}_S[<, +]$, such that $L(\phi)$ is a language with the neutral letter, λ. By Theorem 8.4.8 there exists an active domain sentence $\phi' \in \mathcal{L}_S[<, +, \equiv]$ and a set $\mathcal{R} \subseteq_\infty \mathbb{N}$ such that $\mathcal{R}$ collapses ϕ to ϕ'. Theorem 8.4.9 now gives an active domain formula $\psi \in \mathcal{L}_S[<]$ and an infinite set $Y \subseteq_\infty \mathcal{R}$ such that ψ is Y-weakly equivalent to ϕ'. We now show that $L(\phi) = L(\psi)$. Let $w \in \Sigma^* \lambda^\omega$. Consider the word $w' \in \Sigma^* \lambda^\omega$ got by inserting the neutral letter λ in w in such a way that $\mathrm{nnp}(w') \subseteq Y$. Since $L(\phi)$ is a language with a neutral letter we have that $w \models \phi \Leftrightarrow w' \vDash \phi$. From Theorem 8.4.8 and Theorem 8.4.9 we get $w' \vDash \phi \Leftrightarrow w' \vDash \phi' \Leftrightarrow w' \vDash \psi$. Finally as shown in Lemma 8.4.10, ψ defines a language with a neutral letter and hence $w' \models \psi \Leftrightarrow w \models \psi$. □

8.5 PROOF OF LEMMA 8.4.6

In this section we replace a group quantifier by an active domain formula. Here we make use of the fact that we can a priory restrict our domain as shown in the previous section.

Recall that $\phi = Q_G^m z\langle\phi_1, \ldots, \phi_K\rangle$ and $G = \{m_1, \ldots, m_K, 1\}$. Let $X \subseteq \mathbb{N}$ be an arbitrary infinite set. We show that there exists an infinite set $R_\phi \subseteq X$ and an active domain formula $\phi' \in \mathcal{L}_S[<, +, \equiv]$ such that R_ϕ collapse ϕ to ϕ'. By the assumption of the Lemma, we know that for all $i \leq K$, there exists active domain formulas ϕ_i' such that ϕ_i is weakly equivalent to ϕ_i'. Therefore there exists an infinite set $R_{\phi_1} \subseteq X$ such that for all words w where $nnp(w) \subseteq R_{\phi_1}$ we have that $w \models \phi_1 \Leftrightarrow w \models \phi_1'$. Similarly we can find infinite sets $R_{\phi_K} \subseteq R_{\phi_{K-1}} \subseteq \cdots \subseteq R_{\phi_1} \subseteq X$ such that for all $i \leq K$, and for all words w where $nnp(w) \subseteq R_{\phi_K}$, we have that $w \models \phi_i \Leftrightarrow w \models \phi_i'$. So without loss of generality we assume ϕ_is are active domain formulas. In this section, we will find an active domain formula ϕ' and an infinite set $R_\phi \subseteq R_{\phi_K}$, such that ϕ is weakly equivalent to ϕ'.

Before we go in the details we will give a rough overview of the proof idea. The group quantifier evaluates the product $\prod_j u(j)$, where $u(j)$ is a group element that depends on the set of i such that $w, j \models \phi_i$. So we start and analyze the sets $J_i = \{j \mid w, j \models \phi_i\}$. Since the formulas ϕ_is are active domain formulas, we will see that there are certain positions in the word called "*boundary points*" which are crucial. We see that in between two boundary points, the set J_i is periodic. In the construction of the active domain formula for ϕ we show how to iterate over these boundary points in a strictly increasing order. An active domain quantifier can only iterate over active domain positions, hence we will need nested active domain quantifiers, and a way how to "encode" the boundary points by tuples of active domain positions in a unique and order preserving way. Additionally we need to deal with the periodic positions inside the intervals, without being able to compute the length of such an interval, or even check if the length is zero. Here will make use of the inverse elements that always exist in groups.

We start by analyzing the intervals which occur. Since we consider a fixed set S for the rest of the paper, we will write $\mathcal{L}[<, +]$ for the logic $\mathcal{L}_S[<, +, 0, \{\equiv_q : q > 1\}]$.

8.5.1 *Intervals and Linear Functions*

We first show that every formula ψ with at least one free variable has a normal form. This step is a standard procedure in Presburger's *quantifier elimination* technique [Pre29, End72].

Lemma 8.5.1. *Let $\psi(z) \in \mathcal{L}[<, +]$. Then there exists a formula $\hat{\psi}(z) \in \mathcal{L}[<, +]$ such that ψ is equivalent to $\hat{\psi}$, where all atomic formulas in $\hat{\psi}$ with z are of the form $z > \rho, z = \rho, z < \rho, z \equiv_n \rho$, where ρ is a linear function on variables other than z.*

Proof. Terms in our logic are expressions of the form

$$\alpha_0 + \alpha_1 x_1 + \cdots + \alpha_s x_s \quad \text{, where } \alpha_i \in \mathbb{N}$$

and atomic formulas are of the form

$$\sigma = \gamma, \sigma < \gamma, \sigma > \gamma, \sigma \equiv_m \gamma, c(\sigma)$$

whee σ, γ are linear functions, $c \in \Sigma$ and $m > 1$.
Now using any $M \in \mathcal{S}$, where $m_1 \in M$ is not the identity element, we can rewrite $c(\sigma)$ by an equivalent formula

$$Q_M^{m_1} x \, \neg\lambda(x)\langle (x = \sigma) \wedge c(x), false, \ldots, false\rangle$$

Now consider the atomic formulas containing the free variable z in $\psi(z)$. By multiplying with appropriate numbers, we can re-write these atomic formulas as $nz = \rho, nz < \rho, nz > \rho, nz \equiv_l \rho$ for one particular n, which is the least common multiple (lcm) of all the coefficients in ψ. Here ρ does not contain z and also it might contain subtraction. That is $nz = \rho$ might stand for $nz + \rho_1 = \rho_2$. Now we replace nz by z and conjunct the formula with $z \equiv_n 0$. □

For any formula $\psi(z)$, the notation $\hat{\psi}(z)$ denotes the normal form as in Lemma 8.5.1. Let $x_1, \ldots, x_s$ be the bounded variables occurring in $\hat{\phi}_i(z)$ and $y_1, \ldots, y_r$ be the free variables other than z in $\hat{\phi}_i(z)$. Hence the terms ρ that appear in the formula $\hat{\phi}_i(z)$ can be identified as functions, : $\mathbb{N}^{s+r} \to \mathbb{N}$.

We collect all functions $\rho(\vec{x}, \vec{y})$ that occur in the formulas $\hat{\phi}_i(z)$ for an $i \leq K$:

$$R = \{\rho \mid \text{where } \rho \text{ is a linear term occurring in } \hat{\phi}_i(z), i \leq K\}$$

We define the set T of offsets as a set of terms which are functions using the variables $y_1, \ldots, y_r$ as parameters:

$$T = \{\rho(0, \ldots, 0, y_1, \ldots, y_r) \mid \rho \in R\} \cup \{0\}$$

Consider the set of absolute values of all the coefficients appearing in one of the functions in R. Let $\alpha' \in \mathbb{N}$ be the maximum value among these. That is $\alpha' = max\{|\gamma| \mid f \in R, \gamma \text{ is a coefficient in } f\}$. Let $\Delta = s \cdot \alpha'$. Now we can define our set of extended functions. For a $t \in T$ we define a set of terms which are functions using the variables $x_1, \ldots, x_s, y_1, \ldots, y_r$ as parameters:

$$F_t = \Big\{ \sum_i^{s'} \alpha_i x_i + t \mid s' \leq s, -\Delta \leq \alpha_i \leq \Delta, \alpha_i \in \mathbb{N} \Big\}.$$

We denote by $F = \cup_{t \in T} F_t$.

For a fixed word $w \in \Sigma^* \lambda^\omega$ and a fixed assignment of the free variables $\vec{y}$ to $\vec{d}$ we let

$$B^{w,\vec{d}} = \{f(\vec{d}, \vec{d}) \mid t \in T, f \in F_t, \vec{d} \in \mathrm{nnp}(w)^{s'}, d_1 > d_2 > \cdots > d_{s'}\}$$

be the set of *boundary points*. Note that the assignments to the functions are of strictly decreasing order. Let

$$b_1 < b_2 < \ldots < b_l$$

be the boundary points in $B^{w,\vec{a}}$. Then the following sets are called *intervals*:

$$(0, b_1), (b_1, b_2), \ldots, (b_{l-1}, b_l), (b_l, \infty)$$

Here $(a, b) = \{x \in \mathbb{N} \mid a < x < b\}$ and (b_l, ∞) is called the *infinite interval* . We also split the set of points in $B^{w,\vec{a}}$ depending on the offset

$$B_t^{w,\vec{a}} = \{f(\vec{d}, \vec{a}) \mid f \in F_t, \vec{d} \in \mathrm{nnp}(w)^{s'}, d_1 > d_2 > \cdots > d_{s'}\}.$$

We fix a word $w \in \Sigma^* \lambda^\omega$ and an $\vec{a} \in \mathbb{N}^r$. Therefore we drop the superscripts in $B^{w,\vec{a}}$ ($B_t^{w,\vec{a}}$) and call them B (B_t). Figure 10 illustrates some of the definitions.

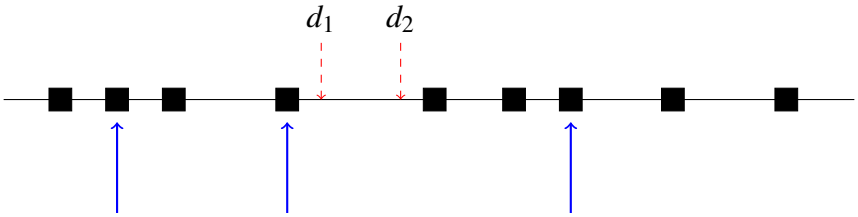

Figure 10: The up arrows point to the *active domain* and the boxes are the *boundary points*, $B^{w,\vec{a}}$. We have marked two points d_1 and d_2 in the same *interval*. All points in an interval have the neutral letter.

The following lemma shows that all points the linear terms in $\hat{\phi}_i$s point to are in B.

Lemma 8.5.2. $\{\rho(d_1, \ldots, d_s, \vec{a}) \mid \rho \in R, d_i \in nnp(w)\} \cup \mathrm{nnp}(w) \subseteq B$

Proof. Let $S = \{\rho(d_1, \ldots, d_s, \vec{a}) \mid \rho \in R, d_i \in nnp(w)\} \cup \mathrm{nnp}(w)$. Since $\rho(x_1) = x_1$ is in $F_0 \in F$, we have $nnp(w) \subseteq B$. Let $b \in S$. Then there is a function $\rho = \sum_i^{s'} \alpha_i x_i + t(\vec{y})$ in F_t and values $p_1, \ldots, p_{s'} \in nnp(w)$ such that $b = \rho(p_1, \ldots, p_{s'}, \vec{a})$. Let $p'_1 > p'_2 > \cdots > p'_l$ be the ordered set of all p_is in the above assignment. We let $\rho'(x_1, \ldots, x_l) = \sum_i \beta_i x_i + t$, where $\beta_i = \sum_{j:p_j = p'_i} \alpha_j$. Therefore $b = \rho'(p'_1, \ldots, p'_l)$. Since $|\beta_i| \leq \Delta \cdot s$ we have $\rho' \in F_t$ and hence $b \in B_t$. □

Let us try to understand the definitions and the lemma seen above. Note that all the quantifiers in $\hat{\phi}_i$s are active domain quantifiers. Thus the bounded variables in $\hat{\phi}_i$s will only be evaluated at the active domain points in w. Now consider a linear term $\rho(x_1, \ldots, x_s, y_1, \ldots, y_r)$ present in one of the $\hat{\phi}_i$s. Observe that once we assign the free variables to $\vec{a}$, then the bounded variables $x_1, \ldots, x_s$ will run over all the active domain points in w. We therefore look at all the points ρ points to, if its bounded variables $x_1, \ldots, x_s$ are assigned values from $nnp(w)$. The importance of these points will be clear once we see Lemma 8.5.3. Lemma 8.5.2 shows that all such points are infact inside the set B. That is B

over-approximates the set of points we want. Lemma 8.5.3 also shows that the over-approximation also preserves the properties we are looking for. Later we will see that this over-approximation is more suitable to build active domain formulas which are weakly equivalent to ϕ.

Let q be the lcm of all q' where $\equiv_{q'}$ occurs in one of the ϕ_is. We need the following lemma, that inside an interval with only neutral letters, the congruence relations decide the truth of an active domain formula.

Lemma 8.5.3. *For the fixed word w and $a_1, \ldots, a_r \in \mathbb{N}$ and let $c, d \in \mathbb{N}$ belong to the same interval in $B^{w,\vec{a}}$ such that $c \equiv_q d$. Then for all $i \leq K$: $w, c \vDash \phi_i(z, \vec{a}) \Leftrightarrow w, d \vDash \phi_i(z, \vec{a})$.*

Proof. Proof is by induction on the structure of the formula $\hat{\phi}_i$ (that is the normal form of ϕ_i). We will now show that $\forall b_i \in \text{nnp}(w)$ and all subformulas $\psi(z, \vec{x}, \vec{y})$ of $\hat{\phi}_i$ that $w, c, \vec{b}, \vec{a} \vDash \psi \Leftrightarrow w, d, \vec{b}, \vec{a} \vDash \psi$. The atomic formulas of $\hat{\phi}_i(z, \vec{a})$ are of the following form: $z < \rho(\vec{x}, \vec{a}), z = \rho(\vec{x}, \vec{a}), z > \rho(\vec{x}, \vec{a}), z \equiv_{q'} \rho(\vec{x}, \vec{a}), a(z)$ and formulas which does not depend on z. It is clear that the truth of formulas which does not depend on z, $a(z)$ and $z \equiv_{q'} \rho$ does not change whether we assign c or d to z. For example, $w, c \models a(z) \Leftrightarrow w, d \models a(z)$, since inside an interval we see only the neutral letter. Let $\vec{b} \in nnp(w)^s$. By Lemma 8.5.2 we know that $\rho(\vec{b}, \vec{a})$ is in B and since c, d lies in the same interval it follows that $c < \rho(\vec{b}, \vec{a}) \Leftrightarrow d < (\vec{b}, \vec{a})$. Similarly we can show that the truth of $z > \rho, z = \rho$ does not change on z being assigned c or d. Thus we have that the claim holds for atomic formulas. The claim clearly holds for conjunction and negation of formulas. Now let the claim hold for subformulas $\psi_1, \ldots, \psi_K$. Therefore $\forall i \leq K$ we have that $\{\vec{b} \in \text{nnp}(w)^s \mid w, c, \vec{b}, \vec{a} \vDash \psi_i\} = \{\vec{b} \in \text{nnp}(w)^s \mid w, d, \vec{b}, \vec{a} \vDash \psi_i\}$. Therefore we have that

$$\begin{aligned} & w, c, b_2, \ldots, b_s, \vec{a} \vDash Q_M^m x \, \neg\lambda(x)\langle\psi_1, \ldots, \psi_K\rangle \\ \Leftrightarrow\ & w, d, b_2, \ldots, b_s, \vec{a} \vDash Q_M^m x \, \neg\lambda(x)\langle\psi_1, \ldots, \psi_K\rangle \end{aligned}$$

And hence it is closed under active domain quantification. □

The above lemma says that the two points d_1 and d_2 in Figure 10 satisfy the same set of formulas ϕ_i if $d_1 \equiv_q d_2$. In other Lemma 8.5.3 says that inside an interval, only the congruence relations can change the satisfiability of the ϕ_is. Thus, it is enough to know the truth values of ϕ_i at a distance of $\geq q$ from the boundary points, since the truth values inside an interval are going to repeat after every q positions. Figure 11 give a pictorial representation.

The following Lemma deals with the infinite interval.

Lemma 8.5.4. *Let b belong to the infinite interval and $\vec{a} \in \mathbb{N}^r$. If $w, \vec{a} \vDash \phi$ then $w, b, \vec{a} \nvDash \phi_i$ for any $i \leq K$.*

Proof. Let $i \leq K$ and b be in the infinite interval and $w, b, \vec{a} \vDash \phi_i$. From Lemma 8.5.3 we know that all points $c \equiv_q b$ and such that c is also in the infinite interval will be witnesses for ϕ_i. This means the set of witnesses is infinite and hence $w, \vec{a} \nvDash \phi$. □

Let us first understand the importance of Lemma 8.5.3 and 8.5.4. Recall from the Preliminaries (Chapter 2) that we denoted by $\Gamma(w,i)$ the group element at position i. That is

$$\Gamma(w,i) = m_j \text{ iff } w, \vec{a} \models \phi_j \wedge \bigwedge_{l<j} \neg\phi_l$$

For a $b \in B$, we define the function $\mathrm{IL}(b)$ to be the length of the interval to the left of b. That is if (b',b) form an interval then $\mathrm{IL}(b) = b - b' - 1$. Similarly we define $\mathrm{IR}(b)$ to be the length of the interval to the right of b. Let $W = q|G|$. We now define two group values.

$$Post(b) = \begin{cases} u(b+1)u(b+2)\dots u(b+\mathrm{IR}(b)) \text{ if } \mathrm{IR}(b) < W \\ u(b+1)u(b+2)\dots u(b+r) \\ \qquad\qquad \text{if } \mathrm{IR}(b) \geq W \text{ and } r < W, b+r \equiv_W 0 \end{cases}$$

$$Pre(b) = \begin{cases} 1_G \qquad\qquad \text{if } \mathrm{IL}(b) < W \\ u(b-r+1)\dots u(b-1) \\ \qquad\qquad \text{if } \mathrm{IL}(b) \geq W \text{ and } r < W, b-r \equiv_W 0 \end{cases}$$

We define

$$N_0(b) = Pre(b).u(b).Post(b)$$

The Figure 11 shows the important points around the boundary points. The following

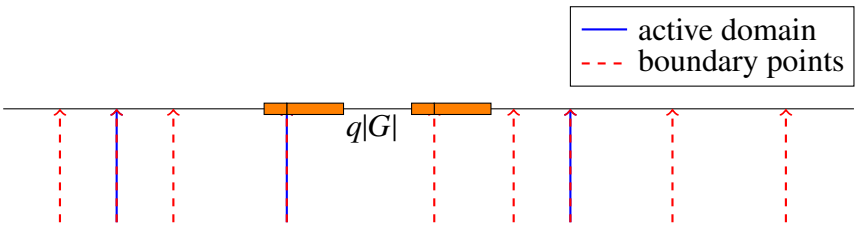

Figure 11: The only points of interest in a word are a fixed area around the boundary points (shaded area shown here). The length between the nshaded area is congruent to $W = q|G|$.

Lemma shows that a finite interval around the boundary points are the "only" points of interest to us.

Lemma 8.5.5. *Let us assume that for all $i \in \mathbb{N}$ in the infinite interval, $\Gamma(w,b) = 1_G$. Then*

$$\prod_{i=1}^{\infty} \Gamma(w,i) = \prod_{i \in B} N_0(i)$$

Proof. Let $b_0 < b_1 < \cdots < b_x$ be the positions in B. Then

$$\prod_{b=b_0}^{b_x} N_0(b) = Pre(b_0)u(b_0)\left(\prod_{i=0}^{x-1} Post(b_i)Pre(b_{i+1})u(b_{i+1})\right)Post(b_x) \tag{7}$$

Now consider an interval (b_i, b_{i+1}). We show that $Post(b_i)Pre(b_{i+1}) = \prod_{j=b_i+1}^{b_{i+1}-1} \Gamma(w,j)$. There are two cases to consider

- Case $b_{i+1} - b_i < W$: Then

$$Post(b_i)Pre(b_{i+1}) = \Big(\Gamma(w, b+1)\Gamma(w, b+2)\ldots\Gamma(w, b+\mathrm{IR}(b))\Big)(1_G) = \prod_{j=b_i+1}^{b_{i+1}-1} \Gamma(w, j)$$

- Case $b_{i+1} - b_i \geq W$: Let $s, t \in \mathbb{N}$, be such that $s, t < W$ and $b_i + s \equiv_W b_{i+1} - t \equiv_W 0$. Lemma 8.5.3 shows that inside an interval all positions congruent modulo q satisfy the same formulas and hence the product of group elements of any $W = q|G|$ consecutive positions evaluate to the identity element. This is because; q consecutive positions evaluate to a group element say m. Since $W = q|G|$ we have that m is generated $|G|$ times and $m^{|G|}$ gives the identity element 1_G (See Preliminaries Chapter 2). Therefore $u(b_i + s + 1)u(b_i + s + 2)\ldots u(b_{i+1} - t) = 1_G$. So

$$Post(b_i)Pre(b_{i+1}) = \Big(\Gamma(w, b_i+1)\Gamma(w, b_i+2)\ldots\Gamma(w, b_i+s)\Big)\Big(\Gamma(w, b_{i+1}-t+1)\ldots\Gamma(w, b_{i+1}-1)$$

Now substituting the value of $Post(b_i)Pre(b_{i+1})$ for all $i < x$ on equation 7 will give us the claim. □

We view $N_0(b)$ as a group value at a point b. Observe that the cardinality of B is finite, even though it might depend on the length of the word w. The following lemma helps us understand when does w be a model of ϕ. Recall that $\phi = Q_G^m z\langle\phi_1, \ldots, \phi_K\rangle$.

Lemma 8.5.6.

$$w, \vec{a} \models \phi \Leftrightarrow \prod_{i \in B} N_0(i) = m \textit{ and for all } b \textit{ in the infinite interval } \Gamma(w, b) = 1_G$$

Proof. $(\Rightarrow)$: Since $w \models \phi(\vec{a})$ we have by Lemma 8.5.4 that for all b in the infinite interval $\Gamma(w, b) = 1_G$. Then the claim follows from Lemma 8.5.5.
$(\Leftarrow)$: Let the right side be true. Then by Lemma 8.5.5 we have that $\prod_i^{\infty} \Gamma(w, i) = m$. The claim now follows from the definition of the group quantifier. □

So it remains to show that there exists an active domain formula which can multiply the group values $N_0(i)$ in the correct order and also that there exists an active domain formula to check whether $u(b) = 1$, for all b in the infinite interval. The major part of the work is in showing the former. For this we need to go through the points in B in an increasing order. The rest of the proof demonstrates

1. How we can treat each B_t differently.
2. There is an active domain formula which goes through the points in B_t in an increasing order

8.5.2 *Treating each B_t differently*

Recall the definition of $N_0(b)$ from the previous section.

Our aim is to give an active domain formula such that the formula evaluates to true iff the group element $\prod_{i=0} \Gamma(w,i)$ is equal to m. The rest of this subsection will be devoted to computing this product in a way which helps in building an active domain formula.

Let $b < b'$ be boundary points in B. Below we compute $\prod_{i=b}^{b'-1} N_0(i)$ in a different way:

$$\prod_{i=b}^{b'-1} N_0(i) = \prod_{i \geq b} N_0(i) \left(\prod_{i \geq b'} N_0(i) \right)^{-1}.$$

Observe that we can compute the product of the interval using two terms that both need to know only one boundary of the interval. It becomes simpler if we note that the two products do not really need to multiply all the elements $\Gamma(w,i)$, for $i \geq b'$ but simply agree on a common set of elements to multiply.

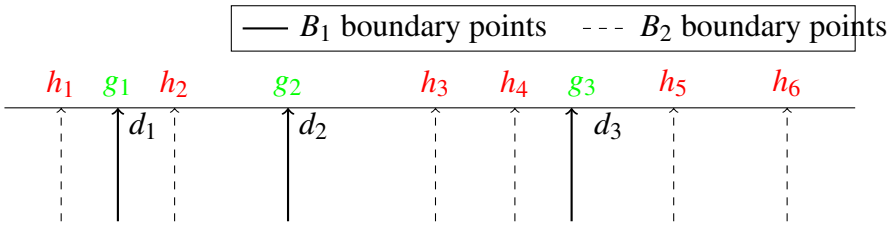

Figure 12: Boundary points and group values there.

The following example will help us understand the method better. Consider Figure 12. The arrows point to boundary points. The **bold** arrows point to the boundary points B_1 and the dashed arrows point to the boundary points B_2. Consider the point marked d_1 and our aim is to compute the product of the group elements to the right of it. We compute the product in the following way.

- n_{d_1} = Product of group elements in $B_2 > d_1 = h_2.h_3.h_4.h_5.h_6$.
- n_{d_2} = Product of group elements in $B_2 > d_2 = h_3.h_4.h_5.h_6$.
- n_{d_3} = Product of group elements in $B_2 > d_3 = h_5.h_6$.
- m_{d_1} = (Group element at d_1)$\times n_{d_1} = g_1.h_2.h_3.h_4.h_5.h_6$.
- $m_{d_2} = n_{d_2}^{-1} \times$(Group element at d_2) $\times n_{d_2} = (h_3.h_4.h_5.h_6)^{-1} g_2 (h_3.h_4.h_5.h_6)$.
- $m_{d_3} = n_{d_3}^{-1} \times$ (Group element at d_3) $\times n_{d_3} = (h_5.h_6)^{-1} g_3 (h_5.h_6)$
- $m_{d_1}.m_{d_2}.m_{d_3}$ gives product of group elements in interval $[d_1, \infty) = g_1.(h_1).g_2.(h_3.h_4).g_3.(h_5.h_6)$.

In the above example we inductively assumed that, starting from any point d_1, we know the product $\prod_{i\in B_2, i>d_1} \Gamma(w,i)$. We show that if we can go through the boundary points in B_1 in an increasing order, then we can compute the product of the group elements at both B_1 and B_2.

We now formalize the above idea. Firstly we define functions which computes the product of the group values for the set B_{t_1}.

$$N_1(b) = \prod_{\substack{b'\in B_{t_1} \\ b'\geq b}} N_0(b')$$

$$\hat{N}_1(b) = \prod_{\substack{b'\in B_{t_1} \\ b'> b}} N_0(b')$$

Note that $N_1(b)$ computes the product of group values at positions greater than or equal to b, whereas $\hat{N}_1(b)$ computes the product of group values strictly greater than b. Inductively we define, for all k, such that $1 \leq k \leq |T|$.

$$N_k(b) = N_{k-1}(b) \prod_{\substack{b'\in B_{t_k} \\ b'\geq b}} (N_{k-1}(b'))^{-1} N_0(b')\hat{N}_{k-1}(b')$$

$$\hat{N}_k(b) = \hat{N}_{k-1}(b) \prod_{\substack{b\in B_{t_k} \\ b'> b}} (N_{k-1}(b'))^{-1} N_0(b')\hat{N}_{k-1}(b')$$

The following lemma shows that $N_k(b)$ computes the product of group values at positions in $\cup_{j\leq k} B_{t_j}$, which are greater than or equal to b. On the other hand $\hat{N}_k(b)$ computes the product of group values at positions in $\cup_{j\leq k} B_{t_j}$ which are strictly greater than b.

Lemma 8.5.7. *Let k be such that $1 \leq k \leq |T|$. Let $b \in \mathbb{N}$. Then*

$$N_k(b) = \prod_{\substack{d\geq b \\ d\in \cup_{j\leq k} B_{t_j}}} N_0(d)$$

$$\hat{N}_k(b) = \prod_{\substack{d> b \\ d\in \cup_{j\leq k} B_{t_j}}} N_0(d)$$

Proof. We prove both the equations by induction over k. The base case, that is when $k = 1$ is true by the definition of $N_1(b)$. So let us assume that the claim is true for $k-1$.

Then, we have for a $b \leq b'$.

$$N_{k-1}(b).\left(N_{k-1}(b')\right)^{-1} = \left(\prod_{\substack{d\geq b \\ d\in\cup_{j<k}B_{t_j}}} N_0(d)\right).\left(\prod_{\substack{d\geq b' \\ d\in\cup_{j<k}B_{t_j}}} N_0(d)\right)^{-1}$$
$$= \prod_{\substack{d=b \\ d\in\cup_{j<k}B_{t_j}}}^{b'-1} N_0(d) \tag{8}$$

For a $b < b'$ we also have that.

$$\hat{N}_{k-1}(b).\left(N_{k-1}(b')\right)^{-1} = \left(\prod_{\substack{d> b \\ d\in\cup_{j<k}B_{t_j}}} N_0(d)\right).\left(\prod_{\substack{d\geq b' \\ d\in\cup_{j<k}B_{t_j}}} N_0(d)\right)^{-1}$$
$$= \prod_{\substack{d>b \\ d\in\cup_{j<k}B_{t_j}}}^{b'-1} N_0(d) \tag{9}$$

We need to now prove the second equality. Let $b \leq b_0 < b_1 < \cdots < b_{x-1} < b_x$ be all positions in B_{t_k}. Writing out the product we get

$$N_k(b) = N_{k-1}(b)\left(N_{k-1}(b_0)\right)^{-1}\left(\prod_{i=0}^{x-1} N_0(b_i)\,\hat{N}_{k-1}(b_i)\left(N_{k-1}(b_{i+1})\right)^{-1}\right)N_0(b_x)\hat{N}_{k-1}(b_x)$$

Substituting the equations 8 and 9 in the above equation gives us the following formula.

$$N_k(b) = \left(\prod_{\substack{d=b \\ d\in\cup_{j\leq k}B_{t_j}}}^{b_0-1} N_0(d)\right)\left(\prod_{i=0}^{x-1} N_0(b_i)\left(\prod_{\substack{d>b_i \\ d\in\cup_{j\leq k}B_{t_j}}}^{b_{i+1}-1} N_0(d)\right)\right)\left(N_0(b_x)\hat{N}_{k-1}(b_x)\right)$$
$$= \prod_{\substack{d\geq b \\ d\in\cup_{j\leq k}B_{t_j}}} N_0(d)$$

Similarly we get that

$$\hat{N}_k(b) = \hat{N}_{k-1}(b)\left(N_{k-1}(b_0)\right)^{-1}\left(\prod_{i=0}^{x-1} N_0(b_i)\,\hat{N}_{k-1}(b_i)\left(N_{k-1}(b_{i+1})\right)^{-1}\right)N_0(b_x)\hat{N}_{k-1}(b_x)$$
$$= \left(\prod_{\substack{d>b \\ d\in\cup_{j\leq k}B_{t_j}}}^{b_0-1} N_0(d)\right)\left(\prod_{i=0}^{x-1} N_0(b_i)\left(\prod_{\substack{d>b_i \\ d\in\cup_{j\leq k}B_{t_j}}}^{b_{i+1}-1} N_0(d)\right)\right)\left(N_0(b_x)\hat{N}_{k-1}(b_x)\right)$$
$$= \prod_{\substack{d>b \\ d\in\cup_{j\leq k}B_{t_j}}} N_0(d)$$

Thus the claim is true for all k. □

Observe that in the above lemma $N_k(b)$ compute the product of group elements at positions greater than or equal to b in the set $\cup_{j\leq k} B_{t_j}$, whereas $\hat{N}_k(b)$ compute the product of group elements at positions strictly greater than b in the set $\cup_{j\leq k} B_{t_j}$.

The following Lemma shows that $N_{|T|}(1)$ gives the product of the group elements.

Lemma 8.5.8. *We have that* $N_{|T|}(1) = \prod_{i\in B} N_0(i)$.

Proof. This follows from Lemma 8.5.7. □

We now give active domain formulas γ^m, $m \in G$, such that γ^m is true iff $N_{|T|}(1) = m$. For this we make use of the inductive definition of N_k and show that there exists active domain formulas γ^m such that $w \models \gamma^m(b) \Leftrightarrow N_k(b) = m$. Similarly we give active domain formulas $\hat{\gamma}^m$ such that $w \models \hat{\gamma}^m(b) \Leftrightarrow \hat{N}_k(b) = m$. Observe that $N_k(b)$ is got by computing the product of $\left(\hat{N}_{k-1}(b')\right)^{-1} u(b') N_{k-1}(b')$, over b', where b' strictly increases. This requires us to traverse the elements in $B_{t_{k-1}}$ in an increasing order. The following section builds a Sorting tree to sort the elements of $B_{t_{k-1}}$ in an increasing order.

8.5.3 *Sorting Tree*

Let $t \in T$. The aim of this section is to create a data structure, which can traverse the elements in B_t in an ascending order. Note that the active domain formulas can only "access" the active domain of the word. But what we need is to access the boundary points. We know that if we assign the variables in a linear term to active domain points, then we can get hold of the boundary point. Thus we can built active domain quantifiers which can iterate through the active domain points of the word, compute the boundary point and generate a group value associated at that particular point. So the first "idea" will be to use active domain quantifiers for all variables appearing in the linear terms. But can we get the active domain points in an ordered way?

The following example will give an intuition of the problem and the solution.

Example 8.5.9. *Consider two linear equations* $2y_1 + y_2$ *and* $y_1 + 5y_2$*, with the* y_i*s being bounded variables. Thus they get assigned from the active domain of the word. Let the active domain of a word* w *be* $D = \{5, 10, 15\}$*. Then the boundary points are*

$$15, 30, 20, 55, 25, 80, 25, 35, 30, 60, 35, 85, 35, 40, 40, 65, 45, 90$$

if we assign the two variables (y_1, y_2) *from* D *in the following order:*

$$(5,5), (5,10), (5,15), (10,5), (10,10), (10,15), (15,5), (15,10), (15,15)$$

Observe that if we follow the particular ordering, then the boundary points, BP is not generated in the order we would want it to be (since it is not in ascending order). We also see that there is repetition of occurrences of boundary points. Thus if we generate

the group values in this ordering and compute its product, we would not get the required product. Since we might be looking at non-commutative groups, we need to ensure that BP is generated in an ascending order.

Here is the solution we propose. We rename the variables and generate the following linear terms:

$$\rho_1 = 2x_1 + x_2, \rho_2 = x_1 + 2x_2, \rho_3 = x_1 + 5x_2, \rho_4 = 5x_1 + x_2$$

The first and third got by replacing y_1 by x_1 and y_2 by x_2. The second and fourth are got by replacing y_1 by x_2 and y_2 by x_1. Now we make use of the fact that we have the leeway to choose the active domain. We will assume that the active domain points we choose are "very far" from each other. Thus, for any assignment to the x_is such that $x_1 >> x_2$ we have

$$5x_1 + x_2 > 2x_1 + x_2 > x_1 + 5x_2 > x_1 + 2x_2$$

Thus we have an ordering for any fixed assignment of the x_is. Now we fix an ordering among different assignments of x_is. Consider Figure 13, where the active domain points are marked r_1, r_2, r_3. Let us look at one of the linear terms, say $\rho_1(x_1, x_2)$. Then we have

Figure 13: The points marked are the active domain points

$$\rho_1(r_3, r_3) > \rho_1(r_3, r_2) > \rho_1(r_3, r_1) > \rho_1(r_2, r_2) > \rho_1(r_2, r_1) > \rho_1(r_1, r_1)$$

since $r_3 >> r_2 >> r_1$. Note that $x_1 \geq x_2$ always. That is we do not consider the points generated by $x_1 < x_2$. But observe that since $\rho_1(x_1, x_2) = \rho_2(x_2, x_1)$ we will be looking at all the boundary points, even if we restrict our variables to this ordering.

We generalize the idea in the above example. In general we need to worry about not two, but a fixed number of linear terms. Also the linear terms might contain variables with negative co-efficients.

To take care of this general situation, we define a tree called *sorting tree*, $\mathcal{T}_t$ which corresponds to the set B_t. The tree satisfies the following property: If the leaves of the tree are enumerated from left to right, then we get the set B_t in ascending order, provided the active domain for the word is choosen judiciously. We will first give the construction of the tree and then the active domain R_ϕ.

Sorting Tree: A node in $\mathcal{T}_t$ is labeled by a tuple (f, A), where $f(x_1, \ldots, x_l)$ is a function in F_t and A an assignment for the variables in f such that $A(x_1) > A(x_2) > \cdots > A(x_l)$ and $\forall i \leq l : A(x_i) \in nnp(w)$.

We show how to inductively built the tree. The root is labeled by the tuple $(t, \{\})$, where t is the function which depends only on $\vec{y}$ (and hence constant on $\vec{x}$) and $\{\}$ is the empty assignment. The root is not marked a leaf node. Consider the internal node $(f(x_1, \ldots, x_l), A)$. It will have three kinds of children ordered from left to right as follows.

1. Left children: These are labeled by tuples of the form (f'_α, A'_j) where

$$f'_\alpha(x_1,\ldots,x_{l+1}) = f(x_1,\ldots,x_l) + \alpha x_{l+1} \text{ and } -\Delta \le \alpha < 0, -\alpha \in \mathbb{N}$$

$$A'_j = A \cup [x_{l+1} \mapsto j], \text{ where } j < A(x_l) \text{ and } j \in \text{nnp}(w)$$

   The left children are now ordered as follows. The tuple $(f'_{\alpha_1}, A'_{j_1})$ is on the left of $(f'_{\alpha_2}, A'_{j_2})$ if $j_1 > j_2$ or if $j_1 = j_2$ and $\alpha_1 < \alpha_2$.

2. Middle child: It is labeled by the tuple (f'', A) where $f''(x_1,\ldots,x_l) = f(x_1,\ldots,x_l)$. It is marked a **leaf** node.

3. Right children: These are labeled by tuples of the form (f'_α, A'_j) where

$$f'_\alpha(x_1,\ldots,x_{l+1}) = f(x_1,\ldots,x_l) + \alpha x_{l+1} \text{ and } 0 < \alpha \le \Delta, \alpha \in \mathbb{N}$$

$$A'_j = A \cup [x_{l+1} \mapsto j], \text{ where } j < A(x_l) \text{ and } j \in \text{nnp}(w)$$

   The children are now ordered as follows. The tuple $(f'_{\alpha_1}, A'_{j_1})$ is on the left of $(f'_{\alpha_2}, A'_{j_2})$ if $j_1 < j_2$ or $j_1 = j_2$ and $\alpha_1 < \alpha_2$.

Observe that if there is no j such that $j < A(x_l)$ and $j \in \text{nnp}(w)$, then (f, A) will only have the single child (f'', A). The tree is built until all functions with s variables appear in leaves and hence the depth of the tree is $s+2$.

Active domain: The infinite set $R_\phi \subseteq R_{\phi_K}$ satisfies the following property: Any two points $a < b$ in R_ϕ is such that $b \ge a(4s\Delta)$. The idea is to ensure that the active domain points are "far" enough to satisfy the nice properties we will soon see. Note that there always exists an infinite set with the above property. We pick any set which satisfies this property and call it R_ϕ.

Let us look at the following example. Figure 14 shows part of a tree, where $\Delta = 2$, $t = 0$ and $\text{nnp}(w) = \{5, 25, 625\}$. Note that the values of the leaves of tree is in ascending order.

Let $\mathcal{R} = 4s\Delta$. Let us also assume that $nnp(w) \subseteq R_\phi$. Given a node (f, A), we say the *value* of the node is the function f evaluated under the assignment of A (denoted by $f(A)$).

Lemma 8.5.10. *Let N be an internal node labeled by a function $f(x_1,\ldots,x_l)$ with $l < s$ and an assignment A. If $A(x_l) = n$, then the children of this node have values in the range $[f(A) - \frac{\Delta}{\mathcal{R}}n, f(A) + \frac{\Delta}{\mathcal{R}}n]$. Moreover the values of the children increases from left to right.*

Proof. The range is given by the construction. Let us look at the tree and the case when both α_1 and α_2 are negative. The other cases are similar to this case or are trivial. There are two cases to consider now. If $j_1 > j_2$, then $\mathcal{R}.j_2 \le j_1$. Therefore $|\alpha_2|.j_2 \le j_1$ and since both the α_is are negative we get $\alpha_2.j_2 > \alpha_1.j_1$, which shows that the value of the children increases from left to right. In the other case, we have that $\alpha_1 < \alpha_2$ and $j_1 = j_2$. Then it is obvious that $\alpha_1.j_1 < \alpha_2.j_2$ and therefore the claim is true. □

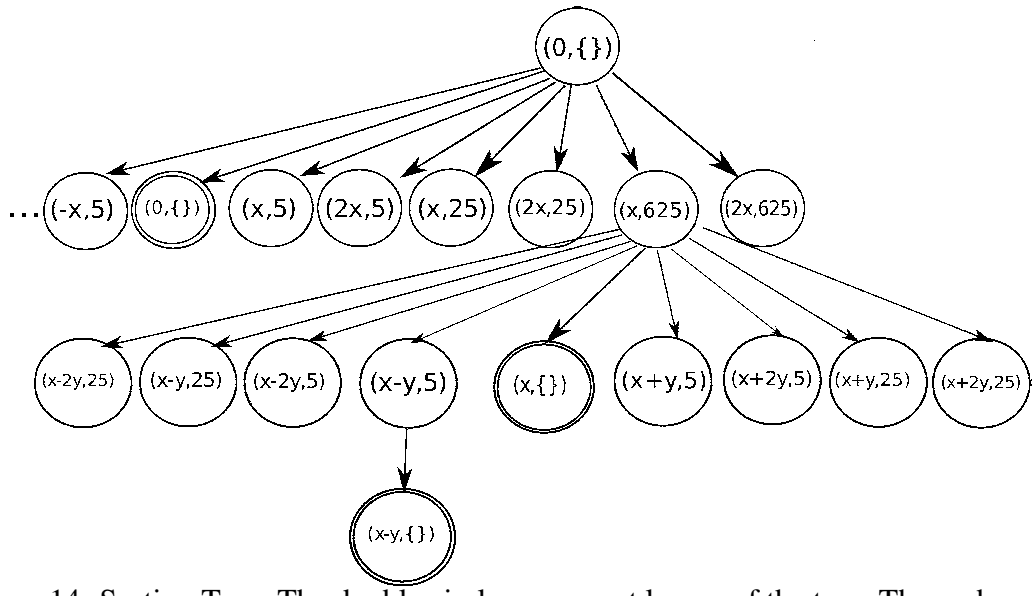

Figure 14: Sorting Tree: The double circles represent leaves of the tree. The nodes of the tree are labeled (f,A), where A is an assignment for the function f and $t=0$. For better presentation we only show the assignment to the newly introduced variable in a node. For example, the tuple $(x-2y,25)$ assigns $x=625$ and $y=25$. The assignment to x is given in the node's parent.

Next we show that for any two neighboring nodes in the tree, the values in the leaves of the subtree rooted at the left node is less than the values in the leaves of the subtree rooted at the right node. Let $V_{(f,A)}$ denote the set of values in the leaves of the subtree rooted at (f,A).

Lemma 8.5.11. *Let (f,A) and (f',A') be neighboring nodes of the same parent such that (f,A) is to the left of (f',A'). Then $u<v$ for every $u\in V_{(f,A)}$ and $v\in V_{(f',A')}$.*

Proof. Let $f=\sum_{i=1}^{l-1}\alpha_i x_i+\alpha_l x_l+t$ and $f'=\sum_{i=1}^{l-1}\alpha_i x_i+\alpha'_l x_l+t$. We show that the rightmost element, u in $V_{(f,A)}$ is less than the left most element, v in $V_{(f',A')}$. From Lemma 8.5.10 and applying induction on the depth of the tree, one can show that $u\leq f(A)+(s-l)\frac{\Delta}{\mathcal{R}}n$ and $v\geq f'(A')-(s-l)\frac{\Delta}{\mathcal{R}}n'$. Let $n=A(x_l)$ and $n'=A'(x_l)$. Let us assume that both coefficients $\alpha'_l,\alpha_l>0$. A similar analysis can be given for other combinations of α'_l and α_l. Now since (f,A) is the left neighbor of (f',A') we have $n<n'$. Then $v-u\geq\alpha'_l n'-(s-l)\Delta\frac{n'}{\mathcal{R}}-\alpha_l n-(s-l)\Delta\frac{n'}{\mathcal{R}}\geq n'-3s\Delta\frac{n'}{\mathcal{R}}>0$. The claim follows, since $\mathcal{R}=4s\Delta$. □

The next lemma says that the values of the leaves of the tree increases as we traverse from left to right.

Lemma 8.5.12. *Let (f,A) and (f',A') be two distinct nodes such that $f(A)<f'(A')$. Then (f,A) appear to the left of (f',A').*

Proof. This follows from Lemma 8.5.11. □

Lemma 8.5.12 shows that if we travel the tree from left to right, then we get the elements in B_t in an ascending order. The following lemma now shows how to build a formula, which can use the sorting tree to traverse through the elements in B_t.

Lemma 8.5.13 (Tree Lemma)**.** *Let $nnp(w) \subseteq R_\phi$. Fix $t \in T$. Assume that for every $b \in B_t$ we have an element $g_b \in G$. For all $f \in F_t$, $m \in G$, $\vec{d} \in nnp(w)^t$, $\vec{a} \in \mathbb{N}^r$, let γ_f^m be active domain formulas such that $w, \vec{d}, \vec{a} \models \gamma_f^m(\vec{x}, \vec{y})$ iff $g_{f(\vec{d},\vec{a})} = m$. Then there are active domain formulas $\Gamma^{m'}$ such that*

$$w, \vec{a} \models \Gamma^{m'}(\vec{y}) \text{ iff } \prod_{b \in B_t} g_b = m'$$

Proof. We will use the sorting tree, $\mathcal{T}_t$ corresponding to B_t for the construction of our formula. Recall that the nodes are labeled by tuples (f, A), where f is a function and A is the assignment of the parameters of f. Let $V_{(f,A)} \subseteq B_t$ be the set of values at the leaves of the subtree rooted at the node labeled by (f, A), and $g_{(f,A)} = \prod_{b \in V_{(f,A)}} g_b$. We will do induction on the depth D of the tree. Let $\tau_f^{m,D}(\vec{x})$ be a formula such that $w, \vec{d} \models \tau_f^{m,D}(\vec{x})$ iff $\prod_{b \in V_{(f,\vec{d})}} g_b = m$ where $(f, \vec{d})$ is the label of a node that has a subtree of depth at most D.

Base Case (leaves): We define $\tau_f^{m,0}(\vec{x}) = \gamma_f^m(\vec{x})$.

Induction Step: Let us assume that the claim is true for all nodes with a subtree of depth at most D. Let the node labeled by (f, A) have a subtree of depth $D+1$. We will need to specify the formula $\tau_f^{m,D+1}(\vec{x})$, where $\vec{x}$ agrees with the assignment A. For every child (f', A') of (f, A) the depth of the corresponding subtree is less than or equal to D. Hence we know we have already formulas by induction.

Recall what the children of (f, A) are: They are of form (f'_α, A'_j) and (f'', A_j). Moreover all nodes (f'_α, A'_j), where α is negative, come to the left of (f'', A_j) and all nodes (f'_α, A'_j), where α is positive, come to its right.

We start by grouping some of the children and computing their product. We let $T^-(A'_j)$ be the product of all subtrees labeled by (f'_α, A'_j) for $\alpha = -\Delta, -\Delta+1, \ldots, -1$. This is a finite product so we can compute this by a Boolean combination of the formulas $\tau_{f'_\alpha}^{m,D}(\vec{x}, x_{l+1})$.

$$\pi_f^{-,m,D}(\vec{x}, x_{l+1}) ::= \bigvee_{m_{-\Delta} \ldots m_{-1} = m} \left(\bigwedge_{\alpha=-\Delta}^{-1} \tau_{f'_\alpha}^{m_\alpha,D}(\vec{x}, x_{l+1}) \right)$$

Now we want to compute the product $\left(\prod_{j \in \text{nnp}(w)} (T^-(A'_j))^{-1}\right)^{-1}$ which is the product of the $T^-(A'_j)$ where $j \in \text{nnp}(w)$ is decreasing. But this can be computed using an active domain group quantifier, $\tau_f^{-,m,D}(\vec{x})$ as follows:

$$\tau_f^{-,m,D}(\vec{x}) = Q_G^{m^{-1}} x_{l+1} \left(\neg\lambda(x_{l+1}) \wedge (x_l > x_{l+1}) \right)$$

$$\langle \pi_f^{-,m_1^{-1},D}(\vec{x}, x_{l+1}), \ldots, \pi_f^{-,m_K^{-1},D}(\vec{x}, x_{l+1})\rangle$$

Recall that the elements of group G are ordered $m_1, \ldots, m_K$. For the single node (f'', A) we already have the formulas $\tau_{f''}^{m,D}(\vec{x})$ by induction (here we have $\vec{x}$ since the assignment A is the same for (f, A) and (f'', A)).

Similarly we define formulas $\pi_f^{+,m,D}(\vec{x}, x_{l+1})$ for the positive coefficients, and compute their product $\prod_{j\in \mathrm{nnp}(w)} T^+(A'_j)$ in an increasing order.

$$\pi_f^{+,m,D}(\vec{x}, x_{l+1}) ::= \bigvee_{m_1 \ldots m_\Delta = m} \left(\bigwedge_{\alpha=1}^{\Delta} \tau_{f'_\alpha}^{m_\alpha,D}(\vec{x}, x_{l+1}) \right)$$

$$\tau_f^{+,m,D}(\vec{x}) ::= Q_G^m x_{l+1} \Big(\neg\lambda(x_{l+1}) \wedge (x_l > x_{l+1}) \Big) \\ \langle \pi_f^{+,m_1,D}(\vec{x}, x_{l+1}), \ldots, \pi_f^{+,m_K,D}(\vec{x}, x_{l+1})\rangle$$

We have now computed the product of the group elements for the three different groups of children. So by a Boolean combination over these formulas we get $\tau_f^{m,D+1}(\vec{x})$:

$$\bigvee_{m'm''m'''=m} \Big(\tau_f^{-,m',D}(\vec{x}) \wedge \tau_{f''}^{m'',D}(\vec{x}) \wedge \tau_f^{+,m''',D}(\vec{x}) \Big)$$

So finally we get $\Gamma^{m'}$ which is same as the formula $\tau_{(t,\{\})}^{m',s+2}$, which is valid at the root of the tree. □

We can also relativize the formulas $\Gamma^{m'}$, with respect to any position.

Lemma 8.5.14. *Let the hypothesis of Lemma 8.5.13 hold. Then for all $m \in G$, there are active domain formulas $\Gamma^m(z, \vec{y})$ such that*

$$w, d, \vec{a} \models \Gamma^m(z, \vec{y}) \text{ iff } \prod_{\substack{b\in B_t \\ b\geq d}} g_b = m$$

Similarly, for all $m \in G$, there are active domain formuals $\Gamma^m(z, \vec{y})$ such that

$$w, d, \vec{a} \models \Gamma^m(z, \vec{y}) \text{ iff } \prod_{\substack{b\in B_t \\ b> d}} g_b = m$$

Proof. We will show how to do the first part of the Lemma. We can conjunct all formulas γ_f^m, for $m \neq 1_G$, with the condition $z \geq f(\vec{x}, \vec{y})$. Whereas the formula γ_f^1, can be disjuncted with the formula $z \geq f(\vec{x}, \vec{y})$. Now applying the tree Lemma 8.5.13 will give us the required formula. □

8.5.4 *Constructing the active domain formula*

This subsection uses all the previous subsections to build the active domain formula we are looking for.

First for all $m \in G, f \in F$, we give formulas γ_f^m such that it is true when N_0 evaluates to m.

Lemma 8.5.15. *For all $m \in G, f \in F$, there are formulas $\gamma_f^m(\vec{x},\vec{y})$ such that for all $d_1,\ldots,d_s \in nnp(w)$, we have that*

$$w \models \gamma_f^m(\vec{d},\vec{a}) \Leftrightarrow N_0(f(\vec{d},\vec{a})) = m$$

Proof. For a $l \in \mathbb{N}$, the following formula checks if there is are points $f'(\vec{x},\vec{y})$ and $f(\vec{x},\vec{y})$ such that the difference between them is l.

$$\delta_f^l(\vec{x},\vec{y}) ::= \bigvee_{f' \in F \setminus f} Q_G^{m_1}\vec{x}'\langle f'(\vec{x}',\vec{y}) = f(\vec{x},\vec{y}) + l, false,\ldots,false\rangle$$

For an $i \leq K$, we denote by $\tilde{\phi}_{m_i}(z,\vec{y})$ the formula $\bigwedge_{j<i}\neg\phi_j(z,\vec{y}) \wedge \phi_i(z,\vec{y})$.

So we have that $\mathrm{IR}(b) = l$ iff $\delta_f^{l+1} \wedge \bigwedge_{l'<l}\neg\delta_f^l$ is true. We define $\pi_f^{m,+l}$ to be true if the product of the first l group elements to the right is m.

$$\pi_f^{m,+l}(\vec{x},\vec{y}) ::= \bigvee_{g_0\ldots g_l = m}\left(\bigwedge_{i=0}^{l}\tilde{\phi}_{g_i}(f(\vec{x},\vec{y})+i,\vec{y})\right)$$

We now give a formula $\psi_{f,m}^{Post}(\vec{x},\vec{y})$ such that $Post(f(\vec{d},\vec{a})) = m$ iff $w \models \psi_{f,m}^{Post}(\vec{d},\vec{a})$. Now we have two cases to consider.

Case $\mathrm{IR}(b) < W$**:** For each of the case $b < b'$ such that $l = b' - b < W$, the formula $\pi_f^{m,l}$ compute the product of the group elements. We define ψ_f^m to be true if the interval is less than W and the product of the group elements is equal to m.

$$\psi_f^m(\vec{x},\vec{y}) ::= \bigwedge_{l=1}^{W-1}\left(\left(\delta_f^l(\vec{x},\vec{y}) \wedge \bigwedge_{l'=1}^{l-1}\neg\delta_f^{l'}(\vec{x},\vec{y})\right) \rightarrow \pi_f^{m,+l}(\vec{x},\vec{y})\right)$$

Case $\mathrm{IR}(b) \geq W$**:** When $b' - b \geq W$ we have to compute the product for the first r group elements, where $b + r \equiv_W 0$ and $r \leq W$. Therefore we define $\hat{\psi}_f^m$ which computes the product of the group elements equal to m in this case.

$$\hat{\psi}_f^m(\vec{x},\vec{y}) ::= \bigwedge_{l=1}^{W}\left(f(\vec{x},\vec{y}) + r \equiv_W 0\right) \rightarrow \pi_f^{m,+r}(\vec{x},\vec{y})$$

A Boolean combination over δ_f^l, $\hat{\psi}_f^m$ and ψ_f^m can give us the formula $\psi_{f,m}^{Post}$.

$$\psi_{f,m}^{Post}(\vec{x},\vec{y}) ::= \left(\left(\bigwedge_{l=1}^{W-1}\neg\delta_f^l\right) \implies \hat{\psi}_f^m\right) \vee \psi_f^m$$

Similarly we give formulas $\psi^{Pre}_{f,m}$ for all $m \in G$ and $f \in F$, such that $Pre(f(\vec{d},\vec{a})) = m$ iff $w \models \psi^{Pre}_{f,m}(\vec{d},\vec{a})$.

For all $m \in G$ and $f \in F$, the formulas, $\gamma^m_f(\vec{x},\vec{y})$ are as follows:

$$\gamma^m_f(\vec{x},\vec{y}) ::= \bigvee_{g_1 g_2 g_3 = m} \psi^{Pre}_{f,g_1} \wedge \tilde{\phi}_{g_2} \wedge \psi^{Post}_{f,g_3}$$

□

We know that for every $b \in B$ there is a function $f \in F$, $d_1, \ldots, d_{s'} \in \mathrm{nnp}(w)$, such that $b = f(\vec{d},\vec{a})$, where $\vec{a}$ is the fixed assignment to the variables $\vec{y}$. We will use this encoding of a position and define a formula $\nu^m_{k,f}$ such that

$$w, \vec{d}, \vec{a} \models \nu^m_{k,f}(\vec{x},\vec{y}) \Leftrightarrow N_k(f(\vec{d},\vec{a})) = m$$

Similarly we define formulas $\hat{\nu}^m_{k,f}$ such that $w, \vec{d}, \vec{a} \models \hat{\nu}^m_{k,f}(\vec{x},\vec{y})$ iff $\hat{N}_k(f(\vec{d},\vec{a})) = m$. We show this by induction over $k \leq |T|$. Starting with the base case $k = 1$.

Lemma 8.5.16. *Let $d_1, \ldots, d_{s'} \in nnp(w)$. For each $m \in G$, there is an active domain formula $\nu^m_{1,f}(\vec{x},\vec{y})$ in $\mathcal{L}[<,+]$, such that if*

$$w \models \nu^m_{1,f}(\vec{d},\vec{a}) \Leftrightarrow N_1(f(\vec{d},\vec{a})) = m$$

Similarly there is an active domain formula $\hat{\nu}^m_{1,f}(\vec{x},\vec{y})$ in $\mathcal{L}[<,+]$ such that if

$$w \models \hat{\nu}^m_{1,f}(\vec{d},\vec{a}) \Leftrightarrow \hat{N}_1(f(\vec{d},\vec{a})) = m$$

Proof. Consider the set B_{t_1} of boundary points formed using functions F_{t_1}. Look at all the formulas γ^m_f, where $f \in F_{t_1}$, given by Lemma 8.5.15. Now we are in a position to apply Lemma 8.5.14, which gives us the required formulas. □

The induction step follows.

Lemma 8.5.17. *Let $d_1, \ldots, d_{s'} \in nnp(w)$. For each $m \in G$, there is an active domain formula $\nu^m_{k,f}$ in $\mathcal{L}[<,+]$, such that*

$$w, \vec{d}, \vec{a} \models \nu^m_{k,f}(\vec{x},\vec{y}) \Leftrightarrow N_k(f(\vec{d},\vec{a})) = m$$

Similarly there is an active domain formula $\hat{\nu}^m_{k,f}(\vec{x},\vec{y})$ in $\mathcal{L}[<,+]$, such that

$$w \models \hat{\nu}^m_{k,f}(\vec{d},\vec{a}) \Leftrightarrow \hat{N}_k(f(\vec{d},\vec{a})) = m$$

Proof. For all $m \in G$ and $f' \in F_{t_k}$ we give formulas $\gamma^m_{f'}$ such that the following holds:

$$w, \vec{d'}, \vec{a} \models \gamma^m_{f'}(\vec{x}, \vec{y}) \Leftrightarrow \left(\hat{N}_{k-1}(b')\right)^{-1} N_0(b')\hat{N}_{k-1}(b') = m$$

Here $d'_1, \ldots, d'_s \in nnp(w)$ and $f'(\vec{d'}, \vec{a}) = b'$. By induction hypothesis there exists formulas $\nu^m_{k-1,f'}$ and $\hat{\nu}^m_{k-1,f'}$ which corresponds to $N_{k-1}(f'(\vec{x}, \vec{y}))$ and $\hat{N}_{k-1}(f'(\vec{x}, \vec{y}))$ respectively. Taking a Boolean combination over these formulas we get the required formula $\gamma^m_{f'}$. We now apply our Tree Lemma 8.5.14 which gives us formulas Γ^m, for all $m \in G$, such that

$$w, \vec{d}, \vec{a} \models \Gamma^m(f(\vec{x}, \vec{y}), \vec{y}) \Leftrightarrow w \models \prod_{\substack{b' \in B_{t_k} \\ b' > b}} \left(\hat{N}_{k-1}(b')\right)^{-1} u(b')N_{k-1}(b') = m$$

Taking Boolean combination over Γ^m and $\nu^m_{k-1,f}$ will give us the formula $\nu^m_{k,f}$. Similarly we can build active domain formulas $\hat{\nu}^m_{k,f}(\vec{x}, \vec{y})$, for all $m \in G$. □

The formulas we were looking for are the active domain formulas $\nu^m_{|T|,f}$, where f is the function which outputs the constant 1. The following Lemma now follows.

Lemma 8.5.18. *Let $nnp(w) \subseteq R_\phi$. Then*

$$\prod_{b \in B} N_0(b) = m \Leftrightarrow w, \vec{a} \models \nu^m_{|T|,1}(\vec{y})$$

Proof. From Lemma 8.5.17 it follows that $w, \vec{a} \models \nu^m_{|T|,1} \Leftrightarrow N_k(1) = m$. The claim now follows from Lemma 8.5.8. □

Now we are in a position to finish the proof of our main Lemma.

Proof of Lemma 8.4.6. By Lemma 8.5.6 we know that it suffices to compute the product of $N_0(b)$, provided the infinite interval evaluate to identity. From Lemma 8.5.18 we know that there are active domain formulas $\nu^m_{|T|,1} \in \mathcal{L}[<, +]$ such that $N_{|T|}(1) = m$ iff $w, \vec{a} \models \nu^m_{|T|,1}$, provided the active domain of the word w comes from the set R_ϕ.

We need to do one last thing. Check that the infinite interval evaluates to 1_G. Replace all formulas $z > \rho$, $z < \rho$, $c(z)$ for a $c \neq \lambda$ and $\lambda(z)$ by *true*, *false*, *false*, *true* respectively in the formulas $\hat{\phi}_i$ and call these formulas $\hat{\psi}_i$. There exists a witness in the infinite interval for the formula $\hat{\phi}_i$ iff $\hat{\psi}_i$ evaluates to true. By Theorem 8.5.4 there should not be any witness in the infinite interval. Therefore the formula $\hat{\psi} = \bigvee_i \psi_i$ evaluates to true iff the infinite interval does not evaluate to 1_G.

Combining both the formulas, we get that $\phi'(\vec{y}) = \neg\hat{\psi}(\vec{y}) \wedge \nu^m_{|T|,1}(\vec{y})$ and therefore for all w, such that $nnp(w) \subseteq R_\phi$ and for all $\vec{a} \in \mathbb{N}^r$ we have that

$$w \models \phi(\vec{a}) \Leftrightarrow w \models \phi'(\vec{a})$$

This completes the proof. □

8.6 DISCUSSION

We have shown that in the presence of a neutral letter the addition relation collapse to linear ordering no matter what monoid quantifier is being used. All languages definable using monoid quantifiers and an order predicate, on the other hand, are regular [BIS90]. Now using semigroup theoretic methods we can separate these classes [Str94]. This enabled us to show separation between various logics which uses addition and order predicates.

Unfortunately if both addition and multiplication are present, then the collapse does not happen. It is also interesting to note that non-solvable groups do not show any surprising property if only addition is present, but as we know from Barrington's theorem non-solvable groups behave quite differently when both addition and multiplication are present.

Figure 15 compare the expressiveness of different logics in the presence of only addition and linear order.

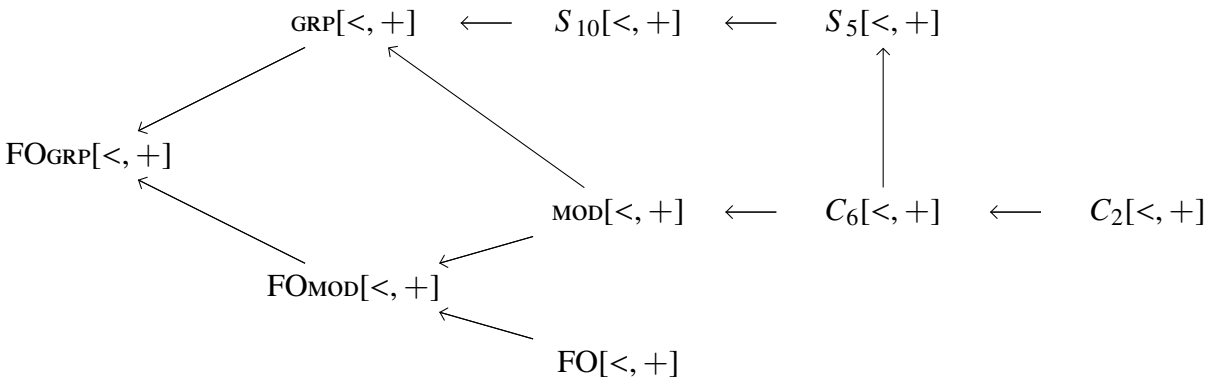

Figure 15: Relation between logics with regular quantifiers and addition relation. The arrows show strict inclusion. No arrow show that the classes are incomparable. C_6 and C_2 denote cyclic groups with 6 and 2 elements respectively and S_{10} and S_5 denote symmetric groups with 10 and 5 elements respectively. The diagram would be similar if the only predicate was $<$. On the other hand, if multiplication is present, the logics behave differently and many questions remain open.

The ultimate objective is to show non-expressibility results for arbitrary predicates or at least when both addition and multiplication are present. The **fundamental question** is to understand

Open Problem 8.6.1. *Is FO*MOD*[<, +, ×] a strict subset of FO*GROUP*[<, +, ×]?*

One possible direction to take this work forward will be to identify the kinds of predicates where our techniques could be applied. That is

Open Problem 8.6.2. *Let S be a subset of monoids. Are there numerical predicates, P such that*

$$\mathcal{L}_{\mathcal{S}}[<,+,P] \cap \mathbf{NLL} = \mathcal{L}_{\mathcal{S}}[<] \cap \mathbf{NLL}?$$

Another way to look at separating the "natural uniform" versions of the complexity classes will be to ask whether one can come up with other suitable restrictions on the set of languages. We saw how restricting the language class to neutral letter languages, help us come up with lower bound results. Can one come up with better restrictions on the set of languages? Inside this restricted set of languages can one show addition and multiplication collapse to order relation? This seems to be the idea Straubing considers in [Str05]. Straubing [Str94] proposes word problems over Regular language as a suitable restriction, while McKenzie, Thomas, Vollmer [MTV10] consider context free languages as a restriction.

Another interesting question which our result fails to answer is whether word problems over non-solvable groups can be defined in $\textsc{maj}[<,+]$ [KLR07]? Can our techniques be of use when working with infinite groups?

Open Problem 8.6.3. *Is $FO\textsc{group}[<,+]$ a subset of $\textsc{maj}[<,+]$?*

9

FOGRP[<, +] SATISFIABILITY

9.1 INTRODUCTION

We first look at the satisfiability of $\mathrm{FO}^2[<, succ, y = 2x, 1]$ and then Presburger arithmetic extended with modulo counting quantifier and finally we look at the satisfiability of the logic $\mathrm{FO}^2{}_{\mathrm{UN}}\mathrm{C}[<, succ]$.

9.2 TWO VARIABLE LOGIC WITH ADDITION

Here we show that the satisfiability of the logic $\mathrm{FO}^2[<, succ, y = 2x, 1]$ is undecidable. We observe first that addition relation in a two variable logic is equivalent to the relation $y = 2x$. Hence we show that the logic $\mathrm{FO}^2[<, +]$ is undecidable.

Theorem 9.2.1. *Satisfiability of $FO^2[<, succ, y = 2x, 1]$ over words is undecidable.*

Proof. We reduce from the undecidability of the *emptiness problem* of *deterministic linear bounded automata* (DLBA) . That is, given a DLBA, M we give a formula $\alpha_M \in \mathrm{FO}^2[<, succ, y = 2x, 1]$ such that

$$\exists w\ M \text{ accepts } w \Leftrightarrow \alpha_M \text{ is satisfiable}$$

Consider a configuration run of M on a word w. Let Γ be the set of all letters which occur in the configuration run. Therefore there exists a mapping $\tau \subseteq (\Gamma^3 \times \Gamma^3)$, which determines for every triple of possible symbols at positions $i-1, i, i+1$, the set of all valid symbols possible at positions $i-1, i, i+1$ in the next configuration. The alphabet for α_M will be $\Sigma = \Gamma^3 \times \mathcal{P}(\{\diamond, \$, \$'\})$ and hence the models of the formula will be a subset of Σ^*. Since the models of the formula correspond to a configuration run of the machine M, we have that every two consecutive triples should satisfy some constraints. That is if $(\gamma_1, \gamma_2, \gamma_3) \in \Gamma^3$ is present at a location i, then the $i+1$th position of the word should satisfy the tuple $(\gamma_2, \gamma_3, \gamma')$, where $\gamma' \in \Gamma$ can be arbitrary. Let us denote this constraint by the "right" relations $\beta \subseteq (\Gamma^3 \times \Gamma^3)$.

We now give an encoding for a model which satisfies the relations β, γ and whch has the final state somewhere. The models for α_M are of the form:

$$\hat{w} = R_0\{\$, \diamond\}C_0\{\$', \diamond\}R_1\{\$, \diamond\}C_1\{\$', \diamond\}R_2...\{\$, \diamond\}C_l\{\$', \diamond\}$$

Here R_0 does not contain the symbols $\diamond$ or \$ or \$'. Infact none of the R_is contain these symbols. Thus the first occurence of $\{\$, \diamond\}$ and $\{\$', \diamond\}$ respectively are unique points in the word. The initial configuration of the run (denoted by C_0 here) lies between these two points. The following conditions can ensure this.

1. There is a unique position, I such that I is the first position where $\{\$, \diamond\}$ is true.
2. There is a unique position, J such that J is the first position where $\{\$', \diamond\}$ is true.
3. $J < 2I$.
4. For all $k \leq I$, the propositions $\diamond, \$, \$'$ are false.

The configuration run of the DLBA consists of the following sequences $C_0C_1...C_l$. Thus the subwords $R_0, R_1, ..., R_{l-1} \in \Sigma^*$ do not have any significance as far as the configuration run is considered. These are "dummy" subwords to help us use our relation $y = 2x$. Even inside C_i for all $i \leq l$, the only points of importance are those points with the $\diamond$. Thus we can verify the final state condition as follows.

5. There exists a position with a $\diamond$ and a final state from Γ.

We ensure that all points in C_0, our initial configuration has the $\diamond$ and thus correspond exactly to an initial configuration of the machine M. Any other point j in C_i contains $\diamond$ if and only if $j = 2k$ and k is a point in C_{i-1} with $\diamond$. We thus propogate the $\diamond$ to all configurations. Similarly we propogate the \$ and \$' symbols. That is j contains \$ symbol if and only if $j = 2k$ (apart from the initial unique point) and k contains the \$ symbol. We list down these conditions below.

6. For all positions k, such that $I \leq k \leq J$, $\diamond$ is true.
7. For all positions $k > J$, $\diamond$ is true at k iff $k = 2l$ and $\diamond$ is true in l.
8. For all positions $k > J$, \$ is true at k iff $k = 2l$ and \$ is true at l.
9. For all positions $k > J$, \$' is true at k iff $k = 2l$ and \$' is true at l.
10. All positions with should have exactly one letter from Γ^3.

Let us denote by γ_k to be the k^{th} letter in $\hat{w}$ restricted to Γ^3. That is $\gamma_k = \hat{w}[k] \cap \Gamma^3$. Because of (10), $\gamma_k \in \Gamma^3$. We next show how the τ relation is maintained in the configurations.

11. Let $2k > J$ has a $\diamond$ and does not have \$ or \$$'$. Then $\tau(\gamma_k, \gamma_{2k})$ should be true.

Now we need to maintain the "right" relation β. For this we also use the points which are inside the configurations C_i but which do not contain the $\diamond$. These points help us to transfer the β relation from a point in C_i with $\diamond$ to the next point in C_i with $\diamond$.

12. Let $2k > J$ has a $\diamond$ and does not have \$ or \$$'$. Then $\gamma_{2k} = \gamma_{2k-1}$ and $\beta(\gamma_{2k}, \gamma_{2k+1})$ holds.

13. Let $\diamond$ be false at k and $k+1$. Then $\gamma_{k+1} = \gamma_k$.

We claim that if we project the letters from Γ^3 in all the positions with the $\diamond$ letter, then we get the configuration run of the machine M.

Claim 9.2.2. *Let $\hat{w}$ be a word which satisfy all the above conditions. Then $\hat{w}_{|\Gamma^3}$ is a successful configuration run of M.*

Proof. The conditions $(1) - (4)$ ensure that the initial configuration is correct. The conditions $(6) - (10)$ ensures that the $\diamond$, \$ and \$$'$ are marked correctly. From the conditions (11), we know that the down relation, τ is respected by the word. The right relations are tranferred from a state with a $\diamond$, along a path with no $\diamond$ to a state with a $\diamond$. Conditions $(12), (13)$ ensure that the right relation is respected. Finally condition (5) ensures that the final state is seen. □

We now show that each of the above conditions can be written in $\mathrm{FO}^2[<, succ, y = 2x, 1]$. In the following formula, x is the unique position which satisfy condition (1):

$$U_1(x) := ((\$(x) \wedge \diamond(x) \wedge \neg\$'(x)) \wedge (\forall y\ (y < x) \implies \neg\$(y) \wedge \neg\diamond(x) \wedge \neg\$'(x))$$

In the following formula, y is the unique position which satisfy condition (2):

$$U_2(y) : (\$'(y) \wedge \diamond(y)) \wedge (\forall x\ (x < y) \implies \neg\$'(x))$$

Using the above formulas we can easily write formulas for conditions $(1) - (5)$. The first row of the input for the machine M, lies exactly between the first and second unique point. In the following formula, x is true iff it points to a position in the first row of the input (between the first and second unique point).

$$FirstRow(x) := \exists y\ (y < x \wedge U_1(y)) \wedge \exists y\ (y > x \wedge U_2(y))$$

Using the above formula we can write Condition (6) as follows: $\forall x\ FirstRow(x) \implies \diamond(x)$. The following formula is true if $x > J$ the second unique position and contains a $\diamond$.

$$ConfigPos(x) := \diamond(x) \wedge \exists y\ U_2(y) \wedge x > y$$

Then Condition (7) (Similarly conditions (8) and (9)) can be written as follows.

$$\forall x\ ConfigPos(x) \Leftrightarrow (\exists y\ (x = 2y) \wedge \diamond(y))$$

Let $\Gamma^3 = \{\gamma_1, \gamma_2, \dots\}$. Condition (10) is easy to state and Condition (11) can be expressed as follows.

$$\bigvee_{i=1}^{|\Gamma^3|} \forall x \ (ConfigPos(x) \wedge \neg\$(x) \wedge \neg\$'(x) \wedge \gamma_i(x)) \Rightarrow \left(\exists y \ (x = 2y) \wedge \bigvee_{(\gamma_j,\gamma_i)\in\tau} \gamma_j(y) \right)$$

Condition (12) can be expressed as follows.

$$\bigvee_{i=1}^{|\Gamma^3|} \forall x \ (ConfigPos(x) \wedge \neg\$(x) \wedge \neg\$'(x) \wedge \gamma_i(x)) \Rightarrow \left(\gamma_i(x-1) \wedge \bigvee_{(\gamma_i,\gamma_j)\in\beta} \gamma_j(x+1) \right)$$

Condition (13) can be written as follows.

$$\bigvee_{i=1}^{|\Gamma|} \forall x \ (\gamma_i(x) \wedge \neg\diamond(x) \wedge \neg\diamond(x+1) \wedge \exists y \ (U_2(y) \wedge y < x)) \Rightarrow \gamma_i(x+1)$$

Conjuncting each of the above $\mathrm{FO}^2[<, succ, y = 2x, 1]$ formula will give us the formula α_M which is satisfiable iff there exists a satisfying run on M. □

9.3 PRESBURGER ARITHMETIC WITH MOD QUANTIFIERS

We now look at Presburger arithmetic [Pre29] [1]. The expressive power of FOMOD over $(\mathbb{N}, <, +)$ is the same as that of FO over $(\mathbb{N}, <, +)$, that is, the *semilinear sets* (Ginsburg and Spanier [GS66]). Infact as Schweikardt et.al [Sch05] points out (see Theorem 7.1.15) Presburger arithmetic is closed under unary counting quantifiers and hence under modulo counting quantifiers.

Let us now look at satisfiability of FOMOD over $(\mathbb{N}, <, +)$. Schweikardt et. al [Sch05] gives a translation from unary counting quantifiers to FO. This translation is non-elementary. Now satisfiability of FO over $(\mathbb{N}, <, +)$ can be done in double exponential space and hence it follows that satisfiability of FOMOD is non-elementary. In this chapter we give a better bound.

Our proof follows the method of Ferrante and Rackoff [FR79], who showed that a quantifier of Presburger arithmetic can be replaced by a bounded quantifier (rather than eliminated straightaway), and the bounding terms are triple exponential in the size of the formula. This gives a nondeterministic 2EXPSPACE machine enough leeway to check for satisfiability. We use the technique to give a 2EXPSPACE upper bound for FOMOD over $(\mathbb{N}, <, +)$. The Ferrante and Rackoff method can also be found in the model theory book [Hod97].

To arrive at this bound, we recall the *Ehrenfeucht-Fraïssé game* [Ehr69, Fra71, Pot94] for modulo counting logic. We call the game corresponding to $\mathrm{FO}_{\mathrm{MOD}}(q)$ as $\mathrm{EF}_{\mathrm{MOD}}(q)$. The

[1] See Chapter Prelimaries 2 for the precise definition and Chapter 7 for properties of this logic.

game is given below. The game starts with two structures $\mathcal{A}$ and $\mathcal{B}$ (with domains A and B, respectively) of the same signature and a number $n \geq 1$. The game consists of n moves, where two players I and II (also called "*spoiler*" and "*duplicator*", respectively) choose elements $(a_i)_{i=1,...,n}$ from A and elements $(b_i)_{i=1,...,n}$ from B, according to the following rules. Before any move, player I decides whether to play the point move or the set move. The two different moves are given below.

Definition 9.3.1. *Point Move*

1. *Player I chooses a structure ($\mathcal{A}$ or $\mathcal{B}$). If I chooses $\mathcal{A}$, I picks an element a_k from that structure. Otherwise I picks an element b_k from the structure $\mathcal{B}$.*
2. *Player II then selects an element from the other structure, so if I chooses $\mathcal{A}$, II picks an element b_k from $\mathcal{B}$. Otherwise II picks an element a_k from $\mathcal{A}$.*

Definition 9.3.2. *Set Move - q-modular move*

1. *Player I chooses $\mathcal{A}$ or $\mathcal{B}$. Assume I chooses $\mathcal{A}$. Player I now picks a set $A_0 \subseteq A$.*
2. *Player II picks a subset $B_0 \subseteq B$, such that $|A_0| \equiv |B_0|(\mod q)$.*
3. *Player I chooses a structure. Assume I chooses $\mathcal{A}$ and picks an element $a_k \in A$.*
4. *Player II picks an element $b_k \in B$, such that $a_k \in A_0$ iff $b_k \in B_0$.*

Player II wins an n-Move game iff after n moves the mapping $(a_1, ..., a_n) \to (b_1, ..., b_n)$ is a partial isomorphism from $\mathcal{A}$ to $\mathcal{B}$, that is, it preserves equality and $<$ and $+$ relation. Otherwise player I wins. We denote by $(a_1, ..., a_k) \sim_q^n (b_1, ..., b_k)$ the fact that player II can always win an n-Move game with the positions $a_1, ..., a_k, b_1, ..., b_k$ initially marked. The equivalence $\sim_q^n$ is called *game equivalence*.

Proposition 9.3.3. *[Pot94] Player II has a winning strategy for a k-Move EF$_{\text{MOD}}(q)$ game on structures $\mathcal{A}$ and $\mathcal{B}$ iff $\mathcal{A}$ and $\mathcal{B}$ satisfy the same FO$_{\text{MOD}}$ formulas over $(\mathbb{N}, <, +)$ of quantifier depth $\leq k$ and whose lcm is q.*

We will use the game to now define an equivalence relation between tuples of numbers "affine equivalence", $\approx_q^n$ which is a finer partition than $\sim_q^n$. We first define sets V_i for all $0 \leq i \leq n$. [2]

Definition 9.3.4.

$$V_0 = \{-2, -1, 0, 1, 2\}$$

$$V_i' = \{\frac{lcm\ V_i}{v}.v' \mid v, v' \in V_i\}, \qquad V_{i+1} = V_i' \cup \{a + b \mid a, b \in V_i'\}$$

[2] See for example Hodges [Hod97] for a corresponding treatment for FO.

Let $N = lcm\ V_n$, $\delta = q^n N^2$. Note that we define *lcm* of a set S to be the smallest positive number which can be divided by all non-zero numbers in S. Now we define linear functions, $F_k^n(\vec{x})$.

Definition 9.3.5.

$$F_k^n(\vec{x}) = \{f(\vec{x}) \mid f = c + \sum_{i=1}^{k} c_i x_i;\ c_i \in V_n \text{ and } c \leq \delta\}$$

Our idea is to show that FOMOD formulas of depth i cannot "represent" any affine function other than those whose coefficients come from V_i, for all i. The following definition makes this idea formal.

Definition 9.3.6. *(affine equivalence) For vectors $\vec{a}$, $\vec{b}$ of length k, $\vec{a} \approx_q^n \vec{b}$ iff for all $f \in F_k^n(\vec{x})$ the conditions below are satisfied.*

$$f(\vec{a}) \leq 0 \text{ iff } f(\vec{b}) \leq 0 \tag{10}$$

$$f(\vec{a}) \geq 0 \text{ iff } f(\vec{b}) \geq 0 \tag{11}$$

$$a_i \equiv b_i (\mod \delta), \forall 1 \leq i \leq k \tag{12}$$

That is if $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$ satisfy the (3) conditions given above, then no FOMOD formula of depth n will be able to distinguish them. The k-ary affine functions expressible using FOMOD formulas of quantifier depth n are in F_k^n.

The following Lemma show that affine equivalence, $\approx_q^0$ refines game equivalence, $\sim_q^0$. That is the lemma shows that if the tuples satisfy the conditions for functions f from F_k^0, then no 0 depth FOMOD formula will be able to distinguish them. This is the base case of our induction.

Lemma 9.3.7. *$\forall k \in \mathbb{N}$ and k length vectors $\vec{a}, \vec{b}$, we have that $\vec{a} \approx_q^0 \vec{b} \Rightarrow \vec{a} \sim_q^0 \vec{b}$*

Proof. Let $\vec{a} \approx_q^0 \vec{b}$. To show that $\vec{a} \sim_q^0 \vec{b}$, it is sufficient to show that $\vec{a}$ and $\vec{b}$ satisfy the same set of formulas of depth 0. The depth 0 formulas are of the form $x_i + x_j = x_l$ and $x_i < x_j$, for all $i, j, l \leq k$. From $\vec{a} \approx_q^0 \vec{b}$ it follows that $\sum_{i=1}^k c_i a_i \leq 0$ iff $\sum_{i=1}^k c_i b_i \leq 0$, for $c_i \in V_0$ and hence the same set of depth 0 formulas are satisfied by $\vec{a}$ and $\vec{b}$. □

The next lemma is the main lemma. It shows that the affine equivalence $\approx_q^n$ is finer than the game equivalence $\sim_q^n$. The lemma also shows that for every tuple $(a_1, \ldots, a_n)$ there exists a tuple $(b_1, \ldots, b_n)$ where all the b_is are "small" and such that $\vec{a} \approx_q^n \vec{b}$.

What does it mean for satisfiability for FOMOD logic? It shows that if a formula is satisfied then we can find "small" numbers which will satisfy the formula. We obtain a double exponential bound on how far one needs to search. Before we begin the lemma recall that $N = lcm\ V_n$, $\delta = q^n N^2$.

Lemma 9.3.8. *For all $q, n \in \mathbb{N}$ and k length vectors $\vec{a}$ and $\vec{b}$; $\vec{a} \approx_q^n \vec{b} \Rightarrow \vec{a} \sim_q^n \vec{b}$. Further there exists a constant, c such that, if every element of $\vec{b}$ is upper bounded by m and $\vec{a} \approx_q^{n+1} \vec{b}$ then for all $a_{k+1} \in \mathbb{N}$ there exists $b_{k+1} \leq q^n m 2^{2^{c(n+k)}}$ such that $\vec{a}, a_{k+1} \approx_q^n \vec{b}, b_{k+1}$.*

Proof. Let $\vec{a}$ and $\vec{b}$ be k length vectors. We prove the claim by induction on n. The base case is given by Lemma 9.3.7.

So let us assume that $\vec{a} \approx_q^{n+1} \vec{b}$ and $\forall k \in \mathbb{N}$ and for all vectors $\vec{z_1}, \vec{z_2}$ of length k and $\forall j \leq n$, we have that $\vec{z_1} \approx_q^j \vec{z_2} \Rightarrow \vec{z_1} \sim_q^j \vec{z_2}$. To prove that $\vec{a} \sim_q^{n+1} \vec{b}$ consider the $n+1$-round game. Suppose Player I chooses a_{k+1}, we will find a b_{k+1} for Player II such that $\vec{a}, a_{k+1} \approx_q^n \vec{b}, b_{k+1}$. By induction hypothesis this will imply $\vec{a}, a_{k+1} \sim_q^n \vec{b}, b_{k+1}$ and hence Player II can always win an $n+1$-round game.

point move: Let Player I pick the element a_{k+1} in structure $\mathcal{A}$. First let us assume that $f(\vec{a}, a_{k+1}) := t(\vec{a}) - ca_{k+1} = 0$, where $f \in F_{k+1}^n$. Then we have that $a_{k+1} = \frac{t(\vec{a})}{c}$. Player II then picks a $b_{k+1} = \frac{t(\vec{b})}{c}$. The three conditions for $\vec{a}, a_{k+1} \approx_q^n \vec{b}, b_{k+1}$ can be proved.

So let us assume now that $f(\vec{a}, a_{k+1}) \neq 0$, for an $f \in F_{k+1}^n$. Consider the set of all statements of the form

$$f(\vec{a}, a_{k+1}) = t(\vec{a}) + ca_{k+1} < 0 \text{ and } f(\vec{a}, a_{k+1}) = t(\vec{a}) + ca_{k+1} > 0$$

where $f \in F_{k+1}^n, c > 0$. Consider the set of terms $H = \{\frac{f(\vec{x})}{c} \mid f \in F_k^n, c \in V_n\}$ and let these terms be ordered as $h_1(\vec{a}) < .... < h_r(\vec{a})$. Let $h_l(\vec{a}) < a_{k+1} < h_{l+1}(\vec{a})$. We show that Player II can pick any b_{k+1} which satisfies:

$$h_l(\vec{b}) < b_{k+1} < h_{l+1}(\vec{b}) \text{ and } a_{k+1} \equiv b_{k+1}(\mod \delta)$$

Let us denote by $g_i(\vec{x}) = Nh_i(\vec{x})$. Since for all $i \leq r$, $g_i(\vec{x}) \in F_k^{n+1}$ and $\vec{a} \approx_q^{n+1} \vec{b}$ we have that for all $i \leq$, $g_i(\vec{a}) \equiv g_i(\vec{b})(\mod q^{n+1} lcm\ V_{n+1})$ and hence $\forall i\ g_i(\vec{a}) \equiv g_i(\vec{b})(\mod N\delta)$. Hence we have:

$$g_l(\vec{a}) < Na_{k+1} < g_{l+1}(\vec{a})$$

We shall now show that there is a b' such that

$$g_l(\vec{b}) < b' < g_{l+1}(\vec{b}) \text{ and } Na_{k+1} \equiv b'(\mod N\delta)$$

If $g_{l+1}(\vec{b}) - g_l(\vec{b}) > N\delta$, then clearly we can find such a b'.

So let us assume that $g_{l+1}(\vec{b}) - g_l(\vec{b}) \leq N\delta$, In that case the following claim gives us that $g_{l+1}(\vec{a}) - g_l(\vec{a}) = g_{l+1}(\vec{b}) - g_l(\vec{b})$ which gives us a b'.

Claim 9.3.9. *Let $f_1(\vec{x}) = d + \sum_{i=1}^k d_i x_i$ and $f_2(\vec{x}) = c + \sum_{i=1}^k c_i x_i$, where the $c_i, d_i \in V_n'$ and $c, d \leq N\delta$. Let $\vec{a} \approx_q^{n+1} \vec{b}$. Then $|f_2(\vec{a}) - f_1(\vec{a})| \leq \delta \Rightarrow f_2(\vec{a}) - f_1(\vec{a}) = f_2(\vec{b}) - f_1(\vec{b})$.*

Proof. Let $\beta = f_2(\vec{a}) - f_1(\vec{a})$ be $\leq \delta$. Now consider the term $g(\vec{x}) = f_2(\vec{x}) - f_1(\vec{x}) - \beta$. We have that $g(\vec{a}) = 0$, that is $(c - d - \beta) + \sum_{i=1}^{k}(c_i - d_i)a_i = 0$. Clearly $(c - d - \beta) \leq 3lcm\ V_n\delta \leq q^{n+1}(lcm\ V_{n+1})^2$ and $(c_i - d_i) \in V_{n+1}$. Thus $g(\vec{x}) \in F_k^{n+1}$ and since $\vec{a} \approx_q^{n+1} \vec{b}$ we have that $g(\vec{a}) = 0$ iff $g(\vec{b}) = 0$ or $f_{i+1}(\vec{b}) - f_i(\vec{b}) = \beta$. □

Player II now picks $b_{k+1} = \frac{b'}{N}$. Tracing back along our argument, we get that (10)–(12) are satisfied.

set move: Let Player I mark a set S in the first word for the q-modular move. Now Player II has to mark a set D. Partition S into intervals $S_0 \cup S_1 \cup ... \cup S_{r+1}$, where $S_0 = S \cap \{h(\vec{a}_k) \mid h \in H\}$ and $\forall i\ 1 \leq i \leq r\ S_i = S \cap \{z \mid h_{i-1}(\vec{a}_k) < z < h_i(\vec{a}_k)\}$ and $S_{r+1} = S \cap \{z \mid z > h_r(\vec{a}_k)\}$. We will show that for each of the S_i's Player II can mark a set D_i such that $|S_i| \equiv |D_i|(\mod q)$ and in addition we have the *congruence conditions* that $\forall a_{k+1} \in S_i, \exists b_{k+1} \in D_i : \vec{a}_k, a_{k+1} \approx_q^n \vec{b}_k, b_{k+1}$ as well as $\forall a_{k+1} \notin S_i, \exists b_{k+1} \notin D_i : \vec{a}_k, a_{k+1} \approx_q^n \vec{b}_k, b_{k+1}$.
For all i such that $h_i(\vec{a}_k) \in S_0$, Player II marks $h_i(\vec{b}_k)$. Thus there is a D_0 with $|D_0| = |S_0|$ and the congruence conditions are satisfied (following from the arguments in Point Move).

Now consider S_i for $i > 0$. We define j-congruence classes

$$S_i^j := \{s \equiv j(\mod \delta) \mid h_i(\vec{a}_k) < s < h_{i+1}(\vec{a}_k)\}$$

Similarly we can define D_i^j in the second word using $\vec{b}_k$. The arguments in the Point Move shows that one can answer an $a_{k+1} \in S_i^j$ by a $b_{k+1} \in D_i^j$. We have $|S_i^j| \equiv |D_i^j|(\mod q)$, since $\forall i \leq r : h_i(\vec{a}_k) \equiv h_i(\vec{b}_k)(\mod q^{n+1}(lcm\ V_{n+1})^2)$. Moreover from Claim 9.3.9 it follows that $|D_i^j| = 0$ iff $|S_i^j| = 0$ and hence we have that $\forall a_{k+1} \in S_i^j, \exists b_{k+1} \in D_i^j : \vec{a}_k, a_{k+1} \approx_q^n \vec{b}_k, b_{k+1}$ and $\forall a_{k+1} \notin S_i^j, \exists b_{k+1} \notin D_i^j : \vec{a}_k, a_{k+1} \approx_q^n \vec{b}_k, b_{k+1}$.

We now come to the second part of the lemma. From the argument above it is clear that we can find a b_{k+1} within a δ-neighbourhood of $F_k^n(\vec{b}_k)$. So

$$\begin{aligned} b_{k+1} &\leq 2\delta + km(Max\ V_n') \leq 2\delta + km(Max\ V_n') \\ &\leq 2q^n(Max\ V_n)^{2|V_n|} + km(Max\ V_n'). \end{aligned}$$

Observe that for all $i \leq n$, $|V_i'| \leq |V_i|^2$ and $|V_{i+1}| \leq 2|V_i'|$. Hence we get $|V_n| \leq 2^{2^{O(n)}}$. Also $Max\ V_{i+1} \leq 2Max\ V_i' \leq 2lcm\ V_i.Max\ V_i \leq 2(Max\ V_i)^{|V_i|+1}$. Hence $Max\ V_n \leq 2^{2^{2^{O(n)}}}$. This gives us that $b_{k+1} \leq q^n m 2^{2^{2^{O(n)}}}$ and hence for all $i \leq k$, $b_i \leq q^n 2^{2^{2^{O(n+k)}}}$. □

The above Lemma gave a tripe-exponential bound on the size of the numbers. Thus we get that

Theorem 9.3.10. *Satisfiability of FO*MOD *over* $(\mathbb{N}, <, +)$ *is in* 2EXPSPACE.

Proof. Let α be an FOMOD formula over $(\mathbb{N}, <, +)$ Lemma 9.3.8 shows that Player II can always place his pebble at a position bounded by $lcm(\alpha)2^{2^{2^{O(n)}}}$. Hence the quantifiers in the FOMOD[+] formula need to be instantiated with values up to this bound. A machine using space $2^{2^{O(|\alpha|)}}$ can run over all the positions and then evaluate a quantifier-free formula with addition. □

Note that binding the quantifier range to $2^{2^{2^{O(n)}}}$ in Presburger arithmetic gives an algorithm ATIME$[2^{2^{O(n)}}, O(n)]$, with the alternations depending on the quantifier alternations in the formula [Ber80]. But since FOMOD also contains modulo quantifiers we do not get this upper bound. The reader will also notice that although EF games have been defined for FOUNC [Ruh99b], our combinatorics is specific to modulo counting and the proof does not easily extend to this logic.

We now show that the satisfiability problem for the two variable logic with unary counting quantifiers is undecidable. This result follows from the fact that $y = 2x$ can be simulated in the two variable logic FO^{2}UNC$[<, succ]$. From our Theorem 9.2.1 we get

Theorem 9.3.11. *Satisfiability of FO^{2}UNC$[<, succ]$ over words is undecidable.*

9.4 DISCUSSION

We first looked at the two variable logic FO$^2[<, y = 2x, succ]$. We showed that the satisfiability of this logic is undecidable. An interesting question left open by our work, concerns the logic without the successor predicate. The undecidability proof given used the *succ* relation. So an interesting question to ask is, Is satisfiability of FO$^2[<, y = 2x]$ undecidable?

Open Problem 9.4.1. *Is the logic FO$^2[<, y = 2x]$ undecidable?*

We then looked at Presburger arithmetic extended with modulo counting quantifiers. Decision procedures for linear arithmetic are closely studied and implemented in many verification engines, see the book by Kroening and Strichman [KS08]. Shankar points out that "even though not many interesting problems are directly expressible in Presburger arithmetic, a great many of the naturally arising proof obligations and subproblems do fall into this decidable class" [Sha02]. The design of specification languages using these quantifiers for verification purposes is an interesting challenge.

Ruhl [Ruh99b] and Schweikardt's [Sch05] papers show that over $(\mathbb{N}, <, +)$, there is an algorithm to convert a FOUNC formula into an equivalent Presburger formula; these papers do not address complexity issues. The complexity of FOUNC over $(\mathbb{N}, <, +)$ (known to be decidable) remains open.

Open Problem 9.4.2. *The complexity of satisfiability of FOUNC over $(\mathbb{N}, <, +)$ is open. Can it be elementarily decidable? A non-elementary decidability exists.*

Part IV

CONCLUSION

10

FUTURE DIRECTIONS AND OPEN QUESTIONS

In this thesis, we looked at logic on words extended with *regular* quantifiers. We studied modulo counting quantifiers and group quantifiers in detail and in different settings.

We first looked at logics which define regular languages like FO[<] and linear temporal logic (LTL) and extended these logics with the above mentioned regular quantifiers. In the second part, we looked at regular quantifiers over a linear order and an addition function which respects the linear order. This took us outside regular languages. For the logics we considered, our primary focus were on the following questions.

1. EXPRESSIVENESS

   We studied the languages/properties definable in these logics. We also saw techniques which are used to prove non-definability of languages by certain logics.

2. SATISFIABILITY

   We also studied the satisfiability of the above logics. In most cases we identified the exact complexity of these problems.

Detailed summary of our results can be found at the end of each chapter. The summaries also give open problems arising out of the work in the respective chapters. The most interesting among these open questions are listed below.

1. The complexity of satisfiability of $\mathrm{FO}^2\mathrm{MOD}(2)[<]$ is left open.
2. Identifying predicates, P such that our techniques can be applied to show that
$$\mathcal{L}_{\mathcal{S}}[<,+,P] \cap \mathbf{NLL} = \mathcal{L}_{\mathcal{S}}[<] \cap \mathbf{NLL}?$$
3. Is $\mathrm{FO}\mathrm{GROUP}[<,+]$ a subset of $\mathrm{MAJ}[<,+]$?
4. Is the logic $\mathrm{FO}^2[<, y = 2x]$ undecidable?
5. The complexity of satisfiability of $\mathrm{FO}\mathrm{UNC}$ over $(\mathbb{N}, <, +)$ is open. Can it be elementarily decidable? Can it be decidable in 2EXPSPACE?

BIBLIOGRAPHY

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[ABO84] Miklos Ajtai and Michael Ben-Or. A theorem on probabilistic constant depth Computations. In ACM, editor, *Proceedings of the sixteenth annual ACM Symposium on Theory of Computing, Washington, DC, April 30–May 2, 1984*, pages 471–474, pub-ACM:adr, 1984. ACM Press.

[Bö60] J. Richard Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundl. Math.*, 6:66–92, 1960.

[Bar89] David A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38(1):150–164, February 1989.

[Ber80] Leonard Berman. The complexity of logical theories. *Theoretical Computer Science*, 11(1):71–77, May 1980.

[BHMV94] Véronique Bruyère, Georges Hansel, Christian Michaux, and Roger Villemaire. Logic and p-recognizable sets of integers. *Bulletin of the Belgian Mathematical Society*, 1(2):191–238, March 1994.

[BIL+05] David A. Mix Barrington, Neil Immerman, Clemens Lautemann, Nicole Schweikardt, and Denis Thérien. First-order expressibility of languages with neutral letters or: The Crane Beach conjecture. *J. Comput. Syst. Sci.*, 70(2):101–127, 2005.

[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, December 1990.

[BKR09a] Christoph Behle, Andreas Krebs, and Stephanie Reifferscheid. Non-solvable groups are not in FO+MOD+MÂJ2[REG]. In *LATA*, pages 129–140, 2009.

[BKR09b] Christoph Behle, Andreas Krebs, and Stephanie Reifferscheid. Regular languages definable by majority quantifiers with two variables. In *Developments in Language Theory*, pages 91–102, 2009.

[BL00a] Michael Benedikt and Leonid Libkin. Expressive power: The finite case. In *Constraint Databases*, pages 55–87, 2000.

[BL00b] Michael Benedikt and Leonid Libkin. Relational queries over interpreted structures. *J. ACM*, 47(4):644–680, 2000.

[BL06] Christoph Behle and Klaus-Jörn Lange. FO[<]-uniformity. In *IEEE Conference on Computational Complexity*, pages 183–189, 2006.

[BMT99] Augustin Baziramwabo, Pierre McKenzie, and Denis Thérien. Modular temporal logic. In *Proc. 14th LICS*, page 344. IEEE, 1999.

[BS91] David A. Mix Barrington and Howard Straubing. Superlinear lower bounds for bounded-width branching programs. In *Structure in Complexity Theory Conference*, pages 305–313, 1991.

[CKK+07] Arkadev Chattopadhyay, Andreas Krebs, Michal Koucký, Mario Szegedy, Pascal Tesson, and Denis Thérien. Languages with bounded multiparty communication complexity. In *STACS*, pages 500–511, 2007.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158, 1971.

[Coo72] David C. Cooper. Theorem proving in arithmetic without multiplication. *Mach. Intell.*, 7:91–99, 1972.

[Dem06] Stéphane Demri. LTL over integer periodicity constraints. *Theor. Comput. Sci*, 360(1-3):96–123, 2006.

[DS02] Stéphane Demri and Philippe Schnoebelen. The complexity of propositional linear temporal logic in simple cases. *Inf. Comput.*, 174(1):84–103, 2002.

[EFT94] H.D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Undergraduate Texts in Mathematics. Springer, 1994.

[Ehr69] A. Ehrenfeucht. An appliation of games to the completeness theorem for formalized theories. *Fund. Math.*, 49:129–141, 1969.

[Eil76] Samuel Eilenberg. *Automata, Languages, and Machines*. Academic Press, New York, 1 edition, 1976.

[Eme90] E. Allen Emerson. Modal and temporal logics. *Handbook of theoretical computer science*, B:995–1072, 1990.

[End72] Herbert B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, New York, 1972.

[EVW02] Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2):279–295, 2002.

[FKPS85] R. Fagin, M. M. Klawe, N. J. Pippenger, and L. Stockmeyer. Bounded-depth, polynomial-size circuits for symmetric functions. *Theoretical Computer Science*, 36(2-3):239–250, April 1985.

[FL79] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. Comp. Syst. Sci.*, 18(2):194–211, 1979.

[FR74] Michael J. Fischer and Michael O. Rabin. Super-exponential complexity of Presburger arithmetic. In R.M. Karp, editor, *Proc. SIAM-AMS Symp. Complexity of computations*, pages 27–41. Amer. Math. Soc., 1974.

[FR79] Jeanne Ferrante and Charles Rackoff. *The computational complexity of logical theories*, volume 718 of *LNM*. Springer, 1979.

[Fra71] R. Fraisse. *Cours de Logique Mathematique: Relation et Formule Logique*. Gauthier-Villars, Paris, 2 edition, 1971.

[FSS84] Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

[Gab87] Dov M. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In *Temporal Logic in Specification*, pages 409–448, London, UK, 1987. Springer-Verlag.

[GOR97] Erich Grädel, Martin Otto, and Eric Rosen. Two-variable logic with counting is decidable. In *12th LICS, Warsaw*, pages 306–317. IEEE, 1997.

[GOR99] Erich Grädel, Martin Otto, and Eric Rosen. Undecidability results on two-variable logics. *Archive for Mathematical Logic*, 38:213–354, 1999.

[GPSS80] Dov Gabbay, Amir Pnueli, Sharanon Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proceedings of the Seventh ACM Symposium on Principles of Programming Languages*, pages 163–173. ACM, 1980.

[GS66] Seymour Ginsburg and Edwin Spanier. Semigroups, Presburger formulas, and languages. *Pacific J. Math.*, 16(2):285–296, 1966.

[Her64] I. N. Herstein. *Topics in Algebra*. Blaisdell, 1964.

[Her75] I. N. Herstein. *Topics in Algebra*. John Wiley & Sons, New York, 2nd edition, 1975.

[Hod97] Wilfrid A. Hodges. *A shorter model theory*. Cambridge, 1997.

[How95] J.M. Howie. *Fundamentals of Semigroup Theory*. London Mathematical Society Monographs. Clarendon, 1995.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, Reading, Mass., 1979.

[Imm86] Neil Immerman. Relational queries computable in polynomial time. *Information and Control*, 68(1–3):86–104, 1986.

[Imm87a] Neil Immerman. Expressibility as a complexity measure: Results and directions. In Stephen R. Mahaney, editor, *Proceedings of the 2nd Annual Conference on Structure in Complexity Theory, CSCT'87 (Cornell University, Ithaca, NY, June 16-19, 1987)*, pages 194–202, Washington, D.C., 1987. IEEE Computer Society, Computer Society Press of the IEEE.

[Imm87b] Neil Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16(4):760–778, August 1987.

[Imm99] Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999.

[JR91] Tao Jiang and B. Ravikumar. A note on the space complexity of some decision problems for finite automata. *Inf. Process. Lett.*, 40(1):25–31, 1991.

[Juk12] Stasys Jukna. *Boolean Function Complexity*, volume 27 of *Algorithms and Combinatorics*. Springer-Verlag, New York, 2012.

[Kam68] Johan A.W. Kamp. *Tense logic and the theory of linear order*. PhD thesis, University of California, Los Angeles, 1968.

[Kle56] S. C. Kleene. Representation of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, editors, *Automata studies*, pages 3–40, Princeton, NJ, 1956. Princeton University Press.

[KLR07] Andreas Krebs, Klaus-Jörn Lange, and Stephanie Reifferscheid. Characterizing TC^0 in terms of infinite groups. *Theory Comput. Syst.*, 40(4):303–325, 2007.

[KPT05] Michal Koucký, Pavel Pudlák, and Denis Thérien. Bounded-depth circuits: separating wires from gates. In *STOC*, pages 257–265, 2005.

[KR65] Kenneth Krohn and John Lewis Rhodes. Algebraic theory of machines. I: Prime decomposition theorem for finite semigroups and machines. *Transactions of the American Mathematical Society*, 116:450–464, 1965.

[Kre08] Andreas Krebs. *Typed semigroups, majority logic, and threshold circuits*. PhD thesis, Eberhard Karls University of Tübingen, 2008. urn:nbn:de:bsz:21-opus-36244; http://d-nb.info/991466683.

[KS08] Daniel Kroening and Ofer Strichman. *Decision Procedures: An Algorithmic Point of View*. Springer, 2008.

[Lan04a] Klaus-Jörn Lange. Some results on majority quantifiers over words. In *IEEE Conference on Computational Complexity*, pages 123–129. IEEE Computer Society, 2004.

[Lan04b] Klaus-Jörn Lange. Some results on majority quantifiers over words. *IEEE Conference on Computational Complexity*, pages 123–129, 2004.

[Lee03] Troy Lee. Arithmetical definability over finite structures. *Math. Log. Q.*, 49(4):385–392, 2003.

[Lib04] Leonid Libkin. *Elements of Finite Model Theory*. Springer-Verlag, Berlin, 2004.

[Lin66] Per Lindström. First order predicate logic with generalized quantifiers. *Theoria*, 32:186–195, 1966.

[LMSV01] Clemens Lautemann, Pierre McKenzie, Thomas Schwentick, and Heribert Vollmer. The descriptive complexity approach to LOGCFL. *J. Comput. Syst. Sci*, 62(4):629–652, 2001.

[LPS10] Kamal Lodaya, Paritosh K. Pandya, and Simoni S. Shah. Around dot depth two. In *Developments in Language Theory*, pages 303–315, 2010.

[LTT06] Clemens Lautemann, Pascal Tesson, and Denis Thérien. An algebraic point of view on the crane beach property. In *CSL*, pages 426–440, 2006.

[Lyn82] James F. Lynch. On sets of relations definable by addition. *J. Symb. Log.*, 47(3):659–668, 1982.

[MTV10] Pierre McKenzie, Michael Thomas, and Heribert Vollmer. Extensional uniformity for boolean circuits. *SIAM Journal on Computing*, 39(7):3186–3206, 2010.

[Myh57] J. Myhill. Finite automata and the representation of events. Technical Report 57-624, WADC, 1957.

[Ner58] A. Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):pp. 541–544, 1958.

[Nur00] Juha Nurmonen. Counting modulo quantifiers on finite structures. *Inf. Comput.*, 160(1-2):62–87, 2000.

[ON80] Hiroakira Ono and Akira Nakamura. On the size of refutation kripke models for some linear modal and tense logics. *Studia Logica: An International Journal for Symbolic Logic*, 39(4):325–333, 1980.

[Opp78] Derek C. Oppen. A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic. *J. Comput. Syst. Sci.*, 16(3):323–332, 1978.

[Pet00] Holger Petersen. Decision problems for generalized regular expressions. pages 22 – 29, 2000.

[Pet02] Holger Petersen. The membership problem for regular expressions with intersection is complete in logcfl. In *19th STACS*, pages 513–522, London,UK, 2002. Springer-Verlag.

[Pin86] J. E. Pin. *Varieties of formal languages*. North Oxford Academic, London, 1986.

[Pnu77a] A. Pnueli. The temporal logic of programs. In *focs77*, pages 46–57, 1977.

[Pnu77b] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57, Providence, Rhode Island, 31 October–2 November 1977. IEEE.

[Pnu77c] Amir Pnueli. The temporal logic of programs. *Proc. 18th Found. Comp. Sci.*, pages 46–57, 1977.

[Pot94] Andreas Potthoff. Modulo-counting quantifiers over finite trees. *Theor. Comput. Sci.*, 126(1):97–112, 1994.

[Pre29] Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. *Comptes Rendus, I. Congrès de Mathématiciens des pays slaves, Warsaw*, pages 192–201, 1929.

[Pri56] Arthur N. Prior. *Time and Modality*. Oxford University Press, Oxford, 1956.

[PV06] Guoqiang Pan and Moshe Y. Vardi. Fixed-parameter hierarchies inside PSPACE. In *LICS '06: Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science*, pages 27–36, Washington, DC, USA, 2006. IEEE Computer Society.

[PW97] J.-E. Pin and P. Weil. Polynomial closure and unambiguous product. *Theory of Computing Systems*, 30(4):383–422, 1997. formerly Mathematical Systems Theory.

[Raz89] A. A. Razborov. On the method of approximations. In ACM, editor, *Proceedings of the twenty-first annual ACM Symposium on Theory of Computing, Seattle, Washington, May 15–17, 1989*, pages 167–176, pub-ACM:adr, 1989. ACM Press.

[Rob49] Julia Robinson. Definability and decision problems in arithmetic. *J. Symb. Log.*, 14(2):98–114, 1949.

[Rob58] Raphael M. Robinson. Restricted set-theoretical definitions in arithmetic. *Proc. Amer. Math. Soc.*, 9:238–242, 1958.

[RR97] Alexander A. Razborov and Steven Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1):24–35, August 1997.

[RS71] R. McNaughton and S. Papert. *Counter-free Automata*. MIT Press, Cambridge, USA, 1971.

[RS07] Amitabha Roy and Howard Straubing. Definability of languages by generalized first-order formulas over $(N, +)$. *SIAM J. Comput*, 37(2):502–521, 2007.

[RT89] John Rhodes and Bret Tilson. The kernel of monoid morphisms. *Journal of Pure and Applied Algebra*, 62(3):227–268, 1989.

[Ruh99a] M. Ruhl. Counting and addition cannot express deterministic transitive closure. In *14th Symposium on Logic in Computer Science (LICS'99)*, pages 326–335, Washington - Brussels - Tokyo, July 1999. IEEE.

[Ruh99b] Matthias Ruhl. Counting and addition cannot express deterministic transitive closure. In *14th LICS*, pages 326–335. IEEE, July 1999.

[RW91] Prabhakar Ragde and Avi Wigderson. Linear-size constant-depth polylog-threshold circuits. *Information Processing Letters*, 39(3):143–146, 16 August 1991.

[SC85] Aravinda Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.

[Sch65] Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Inf. Contr.*, 8:190–194, 1965.

[Sch76] Marcel Paul Schützenberger. Sur le produit de concaténation non ambigu. *Semigroup Forum*, 13:47–75, 1976.

[Sch98] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley Series in Discrete Mathematics & Optimization. John Wiley & Sons, 1998.

[Sch01] Nicole Schweikardt. On the expressive power of first-order logic with built-in predicates, 2001.

[Sch02] Ph. Schnoebelen. The complexity of temporal logic model checking. In Philippe Balbiani, Nobu-Yuki Suzuki, Frank Wolter, and Michael Zakharyaschev, editors, *Advances in Modal Logic*, pages 393–436. King's College Publications, 2002.

[Sch05] Nicole Schweikardt. Arithmetic, first-order logic, and counting quantifiers. *ACM Trans. Comput. Log.*, 6(3):634–671, 2005.

[Ser04] Olivier Serre. Vectorial languages and linear temporal logic. *Theoret. Comp. Sci.*, 310(1-3):79–116, 2004.

[Sha02] Natarajan Shankar. Little engines of proof. In *Proc. 11th Formal Methods Europe, Copenhagen*, volume 2391 of *LNCS*, pages 1–20. Springer, 2002.

[SM73] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time(preliminary report). In *Proceedings of the fifth annual ACM symposium on Theory of computing*, STOC '73, pages 1–9, New York, NY, USA, 1973. ACM.

[Smo87] R. Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In ACM, editor, *Proceedings of the nineteenth annual ACM Symposium on Theory of Computing, New York City, May 25–27, 1987*, pages 77–82, pub-ACM:adr, 1987. ACM Press.

[SN92] J. Stévenne and M. Niézette. An efficient symbolic representation of periodic time. In *Int. Conf. on Information and Knowledge Management, Baltimore*, November 1992.

[SN97] Alexei P. Stolboushkin and Damian Niwinski. y = 2x vs. y = 3x. *J. Symb. Log.*, 62(2):661–672, 1997.

[ST03] Howard Straubing and Denis Thérien. Regular languages defined by generalized first-order formulas with a bounded number of bound variables. *Theory Comput. Syst.*, 36(1):29–69, 2003.

[Sto74] Larry Stockmeyer. *Complexity of Decision problems in Automata and Logic*. PhD thesis, MIT, 1974.

[Str94] Howard Straubing. *Finite automata, formal logic, and circuit complexity*. Birkhauser Verlag, Basel, Switzerland, 1994.

[Str05] Howard Straubing. Inexpressibility results for regular languages in nonregular settings. In *Developments in Language Theory*, pages 69–77, 2005.

[STT95] Howard Straubing, Denis Thérien, and Wolfgang Thomas. Regular languages defined with generalized quanifiers. *Inf. Comput*, 118(2):289–301, May 1995.

[SY00] Kai Salomaa and Sheng Yu. Alternating finite automata and star-free languages. *Theor. Comput. Sci.*, 234(1-2):167–176, 2000.

[Tho97] Wolfgang Thomas. Languages, automata and logic. In *Handbook of formal language theory*, volume III, pages 389–455. Springer, 1997.

[TT02] Pascal Tesson and Denis Thérien. Diamonds are forever: The variety DA. In *Semigroups, Algorithms, Automata and Languages 2001, Proceedings*, pages 475–500. World Scientific, 2002.

[TW98] Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as one quantifier alternation. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC-98)*, pages 234–240, New York, May 23–26 1998. ACM Press.

[Var82] M. Y. Vardi. The complexity of relational query languages. In *Proceedings of the 14th ACM Symposium on Theory of Computing (STOC)*, pages 137–146. ACM Press, 1982.

[Vol99] Heribert Vollmer. *Introduction to circuit complexity*. Springer-Verlag, Berlin-Heidelberg-New York-Barcelona-Hong Kong-London-Milan-Paris-Singapur-Tokyo, 1999.

[vzGG03] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2003.

[WI09] Philipp Weis and Neil Immerman. Structure theorem and strict alternation hierarchy for FO^2 on words. *Logical Methods in Computer Science*, 5(3), 2009.

[WTC61] H. Wang, American Telephone, and Telegraph Company. *Proving Theorems by Pattern Recognition -II*. American Telephone and Telegraph Company, 1961.

INDEX

Symbols

(f, A) 97
(k, k')-type 58
1_M ..... *see* identity element
$B^{w,\vec{a}}$ 88
$B_t^{w,\vec{a}}$ 89
F 88
F_t 88
$L(\phi)$, (language of ϕ) 18, 23
$N_0(b)$ 91
$N_1(b)$ 94
$N_k(b)$ 94
R_{ϕ_K} 87, 98
R_ϕ 87, 98
S_n ..... *see* symmetric group
U_1 10
$[n]$ 8
Δ 88
Γ^Φ 14
Γ_j 15
$\mathbb{Z}$, (integers) 8
MOD, (finite cyclic groups) 11
$\text{MOD}_p[\text{Arb}]$ 72
NLL, (neutral letter languages) 72
$\mathbb{N}$, (natural numbers) 8
Σ, (alphabet) 8
Σ^*, (finite words) 8
Σ^+ 8
Σ^∞, (words) 8
Σ^ω, (right infinite words) 8
$\Sigma_1[\text{Arb}]$ 72
$\approx_q^n$ 112
$\square$ ..... *see* block product
$|w|$ ..... *see* length of word
$|$, $(a|b)$ 8
$\equiv$, $(a \equiv_n b)$ 8
F, (future) 12
$\sim_q^n$ ..... *see* game equivalence
$G_g^F\langle\phi_1, \ldots, \phi_k\rangle$ 14
$G_g^P\langle\phi_1, \ldots, \phi_k\rangle$ 14
$\hat{N}_1(b)$ 94
$\hat{N}_k(b)$ 94
$\hat{\nu}_{k,f}^m$ 103
λ ..... *see* neutral letter
$\ell_{r,q}^F$ 13
$\ell_{r,q}^P$ 13
$\mathcal{B}(\Sigma_1^{0,p})[<]$ 72
$\mathcal{G}$, (finite groups) 11
$\mathcal{L}_\mathcal{S}$ 22, 72
$\mathcal{L}_\mathcal{S}[<, +]$ 73
$\mathcal{L}_\mathcal{S}[<]$ 73
$\mathcal{L}[<, +]$ 87
$\mathcal{M}$ 10
$\mathcal{R}$ 98
$\mathcal{T}_t$ 97
$\models$ ..... *see* satisfies
$Q_M^m x\phi\langle\alpha_1, \ldots, \alpha_K\rangle$ 22
$Q_{U_1}^0$ 22
X, (next) 12
$\not\models$ 20
$\nu_{k,f}^m$ 103
$|w|_a$ 8
P, (past) 12
S, (since) 12
τ-terms 19
true 12
U, (until) 12
Y, (yesterday) 12
bpc, (block product closure) 11
i^{th} letter 8
$u(i)$, (group element at i) 21
$w(i)$ ..... *see* i^{th} letter
Σ_3^P 43
AC^0 9
ACC^0 9
$\text{ACC}^0(p)$ 9
CC^0 9
$\text{CC}^0(p)$ 9

DA 27
ITL ..... seeInterval temporal logic26
LEN^F 13
$\text{LEN}^F(q)$ 13
LEN 13
LEN^P 13
$\text{LEN}^P(q)$ 13
MOD^F 13
$\text{MOD}^F($ ..... $)$13
MOD 13
MOD^P 13
$\text{MOD}^P($ ..... $)$13
NC^1 9
TC^0 9
UITL 28

A

active domain 83
active domain formula 83
addition 19
alternation depth 18
atomic formulas 19

B

Barrington's theorem 77
block product 10
BlockSAT 48
boundary points 89

C

calendar 5
CBC ..... *see* crane beach conjecture
circuits 9, 69
collapse 82
collapse-results 73
congruence conditions 114
congruence relations 19
    unary congruence relations 19
$COUNTSAT(PROP)$ 53
crane beach conjecture 71

D

depth 18, 58
descriptive complexity 70
deterministic linear bounded automata . 107
division by monoid 10
DLBA ... *see* deterministic linear bounded automata
dlogtime uniform circuits 9
dot depth k language 23
duplicator 111
TL[DUR] 53
DUR 14

E

Ehrenfeucht-Fraïssé game 110
emptiness problem 107
empty word 8
equivalent formulas 18

F

Ferrante-Rackoff 75
finite model theory 4
first order logic 19
Fischer-Ladner closure 44
FOGRP 22
$\text{FO}^k[\upsilon]$ 22
FO[<] 20
FOMOD 22
formal system 2
$\text{FO}[\tau]$ 19
$\text{FO}^2[<,+]$ 107
$\text{FO}^2[<, succ, y = 2x, 1]$ 107
free variables 20
future depth 18

G

game equivalence 111
generating set 11
generator of group 11
granularity 5
group 11
    cyclic 11
    non-solvable 11
    permutation 15
    solvable 11
    symmetric 11, 16
group quantifiers 21

H

highly uniform circuit class 73

I

identity element 10
IL(b) 91
impure formula 18
initially equivalent 27
interpretation 20
Interval temporal logic 26
intervals 89
    infinite interval 89
inverse element 11
IR(b) 91

K

Kripke structure 18
Krohn-Rhodes decomposition theorem . 11

L

language 8
length counting operators 13
length of word 8
letter 8
Lindström quantifiers 21
linear bounded automata 10
    deterministic 10
linear order 19
linear temporal logic 12
logic 2
LTL . . . . . *see* linear temporal logic
LTL$_{\mathrm{GRP}}$ 17
LTL$_{\mathrm{GRP}}^{bin}$ 17
LTL$_{\mathrm{GRP}}^{un}$ 17

M

MAJ$[<]$ 22
majority quantifier 21, 73
model 18
model checking problem 18
modulo counting 12
monadic second order logic 25
monoid 10
    DA 11
    aperiodic 11
    solvable 11
monoid quantifier 21

N

natural proof 71
neutral letter 71
neutral letter language 71
nice model 49
$nnp(w)$ . . . . *see* non-neutral letter positions
non-neutral letter positions 80, 83
nonuniform circuits 70

O

operator depth 18
order type 85

P

pad states 49
past depth 18
point move 111, 113
$Post(b)$ 91
$Pre(b)$ 91
Presburger arithmetic 5, 73
presburger arithmetic 23
properties on words 2
propositional symbols 2
propositions 2
pure future formula 18
pure past formula 18
pure present formula 18

Q

q-modular move . . . . . *see* set move
quantified boolean formulas 51
quantifier depth 23
quantifier elimination 74, 87
quantifier free 20

R

R 88
Ramsey 84
Ramsey property 85
recognition by monoid 10
regular expression 8
regular language 8

S

satisfiability problem 18, 23
satisfiable 23

formula *satisfies* word 18
semigroup 10
semilinear sets 110
sentence 20
separated 18, 32
separation property 18, 32
set move 111, 114
small model property 59
Smolensky's result 9, 69
spoiler 111
star free expression 8
star free language 8
successor relation 19
succinct representation of groups 15
syntactic monoid 10

T

T 88
tiling 61
tiling problem 61
    rectangle 61
    square 61
tiling system 61
TL 17
transition system 18
tree lemma 100

U

unambiguous language 9
unambiguous languages 27
unary counting quantifiers 20

V

value of a node 98

W

W 91
weakly equivalent 80
witness index 49
witness state 49
word 8
word problem 10

X

X-weakly equivalent 82