Rheinisch-Westfälische Technische Hochschule Aachen
Lehrstuhl D für Mathematik
Mathematische Grundlagen der Informatik

Diploma Thesis
in
Computer Science

Automata and Growth Functions for the Triangle Groups

Markus Pfeiffer

March 2008

Prof. Dr. Gerhard Hiß
Prof. Dr. Erich Grädel

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, den 26. März 2008

Markus Pfeiffer

Contents

Preface

”There is a theory which states that if ever anybody discovers exactly what the Universe is for and why it is here, it will instantly disappear and be replaced by something even more bizarre and inexplicable.
There is another theory which states that this has already happened.”

(Douglas Adams)

Groups are of central interest in mathematics. Apart from being researched as subject in their own right, they are used as a tool whenever a mathematician wants to formalise symmetries. Groups are also building blocks for other central structures researched in algebra such as rings, fields and modules over these structures.

Already before computers were even built, there were algorithms for groups. Max Dehn gave algorithms to solve the word problem in certain classes of groups in [Deh11]. As soon as computers became availabe to researchers, it was natural to use them for computations that were not feasible before. Even though mathematicians in general do not trust them, computers are a very useful tool that is used extensively today. Most noteably, the classification of the finite simple groups would probably not have been possible without the extensive use of computers.

Today the computer algebra systems such as GAP help when attacking a multitude of problems in the theory of finite groups, if only to rapidly try out computations that would take days when done by hand.. For infinite groups computational methods are not quite as developed. Also infinite objects always come with the extra challenge of representation in a finite computing device. There is a large variety of literature on computational group theory. The book [HEO05] contains good introductions to many basic concepts in computational group theory. These concepts are also implemented in GAP. Apart from that the book gives a wide range of references to more detailed treatments of the introduced concepts.

Obviously the first thing needed is a way to represent groups in an appropriate way on a computer. As computers are only capable to hold a finite amount of data at any time, this representation has to be finite and also efficient. There are many ways to accomplish this, each with different advantages and disadvantages and the most common representations of groups for algorithms are permutations, matrices and finite presentations.

A finite presentations defines a group as a set of strings over a set of generators. These strings are reduced modulo certain relations such that each string uniquely represents one group element.

Regular Languages and **Finite State Automata** are central concepts in computer science. Applications range from compilers to the definition of protocols and to the specification of processes with finitely many states. Usually, formal languages are defined as sets of finite strings over a finite alphabet.

Nevertheless, a formal language may be countably infinite. Regular languages have been characterised as the languages defined by finite state automata, regular expressions, monadic second order logic and also by finite semigroups. The last characterisation gives a hint that there is a connection between algebra and the theory of formal languages. There is an extensive algebraic theory of finite state automata with generalisations to strings of infinite length and trees, which have been introduced by Richard Büchi and Lawrence Landwebe among others.

What is more important for computational group theory is that there are efficient algorithms for finite state automata. Whenever we want to compute something related to a regular language, the viewpoint of a finite state automaton is the most practical. For example, complement union and intersection of regular languages can easily be computed as well as an existential construction, and thus certain first order formulae have an automaton that has exactly the language of strings that make the formula true.

Automatic Groups are at the boundary between formal languages and automata and algebra. Automatic groups were introduced by John Cannon and David Epstein and the basic theory has been published in the book [EPC⁺92]. When a group is given as a finite presentation it is thought of as a set of strings. One fundamental problem of finite group presentations is that in most cases it is very hard to tell for a human if two strings represent the same group element. Seemingly very simple tasks such as determining the order of the following finitely presented group are not trivial.

$$G = \langle\, x,y,z,t \mid x^3y^7, y^4x^7, z^3t^5, t^4z^6, [x,z] \,\rangle$$

Computers can help in this situation. For finite groups we can for example try to enumerate all group elements or try orbit stabiliser algorithms to compute the order of the group. Unfortunately the word problem, that is determining whether a string represents the identity, is undecidable in general.

It is a very nice feature of automatic groups that the set of strings that represent group elements is a regular language and that we can multiply and invert group elements using finite state automata. We can also algorithmically tell whether two strings represent the same element of the group. Thus we have a connection between the theory of finite state automata and the theory of finitely presented groups, which enables us to use the algorithmic tools of finite state automata for computations in groups, most noteably also in infinite groups. Also, Derek Holt implemented the algorithms given in [EPC⁺92] and we are able to practically use these methods.

Inspired by automatic groups, Khoussainov and Nerode in [KN95] introduced the concept of **Automatic Structures**. Automatic structures are a generalisation of the concept of an automatic group to algebraic structures. The theory has grown into an active area of research on its own. This is because the class of automatic structures is a very robust class in the sense of logic characterisations and complexity theory. Many basic algebraic structures have been shown to allow for automatic presentations. Blumensath and Grädel developed the theory of automatic structures in [Blu99], [BG04] and following papers.

This thesis deals with an infinite class of automatic groups, the *triangle groups*. The triangle groups are a class of mostly infinite groups with a very simple finite presentation. Although there are algorithms that are able to compute automatic presentations for automatic groups, the detailed structure of such presentations is unknown. Also, these algorithms are rather limited. Thus it is interesting to look at tractible examples and try to describe automatic presentations of triangle groups as good as possible.

Contents

The first chapter introduces the theory of finite state automata as well as the theory of finitely presented groups. Also, there is an exposition of how finite state automata are relevant in the context of algebraic structures.

This leads up to the second chapter, which introduces how finite state automata can be used to define a finitely presented group and gives a few properties that a finitely presented group has to possess to allow for such a presentation. The most important point is that there is a set of axioms that expresses, given a finitely presented group and a set of automata, whether the automata form an automatic presentation for the group.

The third chapter introduces algorithms for automatic groups. We show how the word problem can be solved for such groups in time quadratic in the length of the input. We also show how we can compute an automatic presentation for a finitely presented group. The last section of the third chapter contains some exploration of the connection between Nerode congruence classes of the language of the word acceptor of a group and certain strings that represent elements of the group that are close to the identity element of the group.

In the fourth chapter we specialise to the triangle groups. Triangle groups can be defined as groups of translations and rotations of a geometric space that fix a tiling of this space by triangles. We show how to construct a word acceptor for a triangle group. We also explore what happens when the set of generators is changed. As an application we compute growth functions for some of the triangle groups.

The fifth chapter gives a few conjectures that generalise findings in the fourth chapter.

In the last chapter we give a short summary of the results obtained from the computations as well as suggesting further questions that might be worth answering.

Acknowledgment

I would like to thank Professor Dr. Gerhard Hiß who guided me through the process of writing this thesis and Professor Dr. Erich Grädel who introduced me to the subject of automatic groups. I also thank them for reviewing this thesis. I would also like to thank Professor Dr. Derek Holt who had the idea for the topic of this thesis.

I thank my parents Wilhelm Pfeiffer and Ortrud Pfeiffer and my sister Anne-Catrin and my brother Martin Pfeiffer for their support.

I thank my friends Henning Begerau, Marc Aschmann, Melanie Weidenbörner and Daniel Bauer for their constant support.

Finally, I thank Julia Bräutigam for just being there at the right time in the right place and whenever I needed her.

1 Basic Theory

”Here we present the mathematical theory of such structures. Clearly these fundamentals will stay with us; we are to live with them, and so they had better be elegantly (i.e. simply) defined.”

(Richard J. Büchi)

In this first chapter we will collect some basic notions and theorems that will be used throughout this thesis. This is particularly important because of the diversity of literature used in the process of writing it. The references reach from algebra, the theory of monoids and groups over mathematical logic to the theory of computation where finite state automata are one of the most important tools for us.

Notation varies across fields and authors, thus we will fix a consistent one for our purposes. We will keep a short list of the most important notations used in Appendix A.

The first section contains some introduction to decidability. After that there is a section about algebraic structures and first order logic. This is followed by the theory of semigroups and monoids. We introduce formal languages and the notion of regular languages, as well as finite state automata and the theoretic tools we need in later sections. The section concludes with the introduction of autmatic presentation for algebraic structures. This indicates how finite state automata are relevant in the theory of algebraic structures.

The chapter is concluded with a section about groups. Groups are important enough to justify a section of their own. Apart from that there a some ideas of geometric group theory that would not fit into the section about semigroups and monoids.

Readers who feel familiar with the aforementioned theory are free to read on in Chapter 2 and only come back to this chapter for reference.

1.1 Computability and Decidability

In this section we will shortly deal with computability and decidability. There are often misconceptions about the concept of decidability, even among computer scientists. However, this section does not aim at being a complete introduction to computabiliy of functions.

The mathematician Alan Turing defined his Turing machine as a concept to describe a mechanically working mathematician. The definition of a Turing machine is astonishingly simple. It is usually thought of as an infinite tape together with a head that can read from the tape and write to it at the current position and is controlled by a finite state control.

Definition 1.1 (Turing Machine):
We define a non-deterministic Turing machine *as a* $7-$*tuple* $(Q, A, B, q_0, q_{accept}, \Box, \tau)$, *with*

- *A finite set of states Q.*
- *An alphabet A of input symbols.*
- *An alphabet $B \supset A$ of working symbols, containing $\square$, the blank symbol.*
- *The transition relation $\tau \subseteq (Q \times B) \times (Q \times B \times \{R, L, N\})$.* $\square$

The transition relation might need a bit of explanation. If a tuple (q, a, q', a', D) is an element of τ this means that the machine, when currently in state q reading the symbol a on the tape, can change into the state q' writing the symbol a' onto the tape at the current position and then moves the head to either the left or right or not move it at all, depending on D. Because we defined a non-deterministic Turing machine, there might be other tuples (q, a, q'', a'', D') in τ.

To describe the full state information of a Turing machine, we need to keep track of the contents of the tape, the head position and the current state. This information is usually called a *configuration*. An *initial configuration* consists of the input by printed on the tape and state q_0 with the head positioned at the beginning of the tape.

A *computation* of a Turing machine consists of a sequence of configurations starting with an initial configuration and such that a configuration C' follows a configuration C if there is a transition in τ that makes C into C'.

A computation is *accepting* if the Turing machine reaches a configuration containing the state q_{accept} after a finite amount of time.

We call a function $f : \mathbb{N} \to \mathbb{N}$ *computable* if and only if there is a Turing machine that computes f. That is, given a natural number n as input, the Turing machine halts and gives $f(n)$ as output on the tape. Usually we use binary representation for natural numbers in this context.

The Turing machine is a universal computational model in the sense that every other model of computation that has been thought of today can be simulated by a Turing machine. The *Church-Turing thesis* conjectures that the Turing machine is *the* model of computation in the sense that any function that is naturally regarded as being computable is computable by a Turing machine.

Sometimes we want to decide whether a mathematical object has a certain property, for example whether a group is abelian or whether it is the trivial group. Questions that allow for a simple **Yes** or **No** answer are called *decision problems*. Decision problems can be seen as functions $f : \mathbb{N} \to \{0, 1\}$ if we have an appropriate way to encode our questions in natural numbers. It was shown by Gödel that this is always possible. We call a decision problem *d*ecidable if the function f as given above is computable by a Turing machine. In particular, we call subsets of the natural numbers decidable if their characteristic function is computable.

The *classical decision problem* is the question whether a given first-order formula is satisfiable. The classical decision problem is often also called the “Entscheidungsproblem”, because David Hilbert called it that way in his programme on mathematics.

There are quite innocent sounding problems that are undecidable. For example it is undecidable in general, whether a Turing machine halts on any given input. An undecidable decision problem that is relevant for this thesis is the word problem for groups, which will be introduced in Section 1.5.

Some undecidable problems allow for an enumeration of n for which $f(n) = 1$ holds. These problems are called semi-decidable or recursively enumerable. One of the easiest examples for this are the Turing machines that stop their computation after finitely many steps. This is because we can in fact make a list of encodings of Turing machines and let another Turing machine simulate one step at a time via a diagonal trick. This machine can then output the encodings of the machines that halted after a finite amount of steps. It can not decide whether some Turing machine will not halt.

Also, if a problem is undecidable in general, there often are restricted cases of the problem for which we actually can give a decision procedure. This often leads to confusion. The word problem for groups is undecidable in general, however in this thesis we will give a decision procedure for the word problem for the class of automatic groups.

1.2 Algebraic Structures and First Order Logic

The notion of an algebraic structure is the most basic notion in algebra and it is sufficient to capture almost all important objects studied in mathematics and theoretical computer science.

Let A be a set. Defining $A^0 := \{\emptyset\}$ and $A^{n+1} := A^n \times A$ for $n \in \mathbb{N}$, an n-ary relation R is a subset of A^n and an n-ary function f is a map from A^n to A, denoted by

$$f : A^n \to A \, : \, (a_1, \cdots, a_n) \mapsto f(a_1, \cdots, a_n)$$

The special case of a nullary function denotes a constant. Function application will be written to the left of the argument and will have precedence over any other operation.

We fix a set τ of symbols consisting of relation symbols and function symbols, each equipped with a finite arity. We call τ a *signature*. A signature contains names which will be given to relations and functions and will play an important role when defining logic formulae.

A τ-*structure* $\mathfrak{S}$ consists of a set S, also called its universe, and a map that associates every relation symbol with a relation on S and every function symbol with a function on S, each of appropriate arity. It is important to distinguish between symbols of a signature and the realisation of these symbols in a certain structure. To make clear what we are talking about, we denote elements of a signature by lowercase latin letters for functions and uppercase latin letters for relations. When talking about a structure $\mathfrak{S}$ and functions and relations in this structure we add a superscript to the symbols. For example if f is a function symbol and $\mathfrak{S}$ is an algebraic structure whose signature contains f, then $f^{\mathfrak{S}}$ denotes the function on S.

A τ-substructure $\mathfrak{T} \leqslant \mathfrak{S}$ is a subset T of S that is closed under all relations and functions.

Structures whose signature only contains relations are called relational structures and every algebraic structure can be associated a structure, whose signature only contains relation symbols. This is done by replacing every function with its graph, which for an n-ary function f is defined as

$$G_f := \{(a_1, \cdots, a_n, f(a_1, \cdots, a_n)) \mid (a_1, \cdots, a_n) \in A^n\}.$$

We will denote algebraic structures by uppercase gothic letters $\mathfrak{U}, \mathfrak{V}, \mathfrak{W}$ and their universes by the corresponding latin uppercase letter U, V, W. Relations will be denoted by uppercase latin letters like R, S, T and functions by lowercase latin letters like f, g, h except for the cases where there is a common mathematical notation for 2-ary operators, like multiplication, which will be denoted by $a \cdot b$ or simply written as ab.

Given a signature τ we form first order formulae. For this we need a countable set VAR of variables denoted by $x_1, x_2, \ldots$.

We first form terms.

Definition 1.2 (term):
The set of τ-terms is inductively defined as follows.

- *Every variable x_i is a term.*
- *If $t_1, \cdots, t_n$ are τ-terms and f is an n-ary function symbol in τ then $f(t_1, \cdots, t_n)$ is also a τ-term.*□

Using terms we define the set $\mathrm{FO}[\tau]$ of first order formulae in the signature τ.

Definition 1.3 ($\mathrm{FO}[\tau]$ **formula):**
Let τ be a signature. We inductively define the set $\mathrm{FO}[\tau]$ of first order formulae.

- *If t_1 and t_2 are τ-terms then $t_1 = t_2$ is a $\mathrm{FO}[\tau]$ formula.*
- *If $t_1, t_2, \cdots, t_n$ are τ-terms and R is an n-ary relation symbol, then $R(t_1, \cdots, t_n)$ is a $\mathrm{FO}[\tau]$ formula.*
- *If φ is a $\mathrm{FO}[\tau]$ formula, then $\neg\varphi$ is also a $\mathrm{FO}[\tau]$ formula.*
- *If φ and ψ are $\mathrm{FO}[\tau]$ formulae, then $\varphi \vee \psi$, $\varphi \wedge \psi$ and $\varphi \rightarrow \psi$ are also $\mathrm{FO}[\tau]$ formulae.*
- *If φ is a $\mathrm{FO}[\tau]$ formula and x is a variable then $\exists x \varphi$ and $\forall x \varphi$ are also $\mathrm{FO}[\tau]$ formulae.* □

Quantors *bind* variables. An occurrence of a variable in a fomula is *free* if it is not bound by a quantor. To actually determine the truth value of a given first order formula we have to interpret it in a τ-structure.

Definition 1.4 (interpretation):
A τ-interpretation $\mathfrak{I}$ is a pair $(\mathfrak{S}, \beta)$, where $\mathfrak{S}$ is a τ-structure and β is a partial map that assigns values from S to variables. □

An interpretation assigns values from S to terms and truth values to formulae via the following definition. We say that we define the semantics of terms and formulae.

Definition 1.5 (semantics of $\mathrm{FO}[\tau]$ **formulae):**
Let φ be a $\mathrm{FO}[\tau]$ formula and $\mathfrak{I} = (\mathfrak{S}, \beta)$ be an interpretation. As truth values we use $0 \in \mathbb{N}$ for false and $1 \in \mathbb{N}$ for true. We inductively define the semantics of terms and $\mathrm{FO}[\tau]$ formulae.

Semantics for terms.

- $[\![x]\!]^{\mathfrak{I}} := \beta(x)$
- $[\![f(t_1, \ldots, t_n)]\!]^{\mathfrak{I}} := f^{\mathfrak{S}}\left([\![t_1]\!]^{\mathfrak{I}}, \cdots, [\![t_n]\!]^{\mathfrak{I}}\right).$

Semantics for formulae.

- $[\![t_1 = t_2]\!]^{\mathfrak{I}} := \begin{cases} 1 & \text{if } [\![t_1]\!]^{\mathfrak{I}} = [\![t_2]\!]^{\mathfrak{I}} \\ 0 & \text{otherwise} \end{cases}$
- $[\![R(t_1, \cdots, t_n)]\!]^{\mathfrak{I}} := \begin{cases} 1 & \text{if } \left([\![t_1]\!]^{\mathfrak{I}}, \cdots, [\![t_n]\!]^{\mathfrak{I}}\right) \in R^{\mathfrak{S}} \\ 0 & \text{otherwise} \end{cases}$
- $[\![\neg\varphi]\!]^{\mathfrak{I}} := 1 - [\![\varphi]\!]^{\mathfrak{I}}$
- $[\![\varphi \vee \psi]\!]^{\mathfrak{I}} := \max\left\{[\![\varphi]\!]^{\mathfrak{I}}, [\![\psi]\!]^{\mathfrak{I}}\right\}$
- $[\![\varphi \wedge \psi]\!]^{\mathfrak{I}} := \min\left\{[\![\varphi]\!]^{\mathfrak{I}}, [\![\psi]\!]^{\mathfrak{I}}\right\}$

- $[\![\exists x\varphi]\!]^{\mathfrak{I}} := \max_{s\in S} [\![\varphi]\!]^{\mathfrak{I}[x/s]}$
- $[\![\forall x\varphi]\!]^{\mathfrak{I}} := \min_{s\in S} [\![\varphi]\!]^{\mathfrak{I}[x/s]}$

Where $[x/s]$ means that we replace every free occurrence of x in φ by the element s in S.

Note that $\wedge$, $\forall$ and $\rightarrow$ are redundant, as $\varphi \wedge \psi \equiv \neg(\neg\varphi \vee \neg\psi)$, $\forall\varphi \equiv \neg\exists\neg\varphi$ and $\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$, but for better readability these are included in the syntax for the formulae. In some cases proofs can be shortened because it suffices to give a proof for the reduced set of operations only. □

We say that a structure $\mathfrak{S}$ is a *model* of a formula φ, denoted by $\mathfrak{S} \models \varphi$, if there is an interpretation $\mathfrak{I} = (\mathfrak{S}, \beta)$ such that $[\![\varphi]\!]^{\mathfrak{I}} = 1$.

For all further treatment we refer the reader to an introductory book on first order logic. Mathematical logic is in fact a very interesting and useful subject in its own right and sometimes comes very close to algebra.

We conclude this section with examples of algebraic structures: orders. An order gives a means to compare elements of the universe and choose, for example, more favorable ones. As we will need the notion of an order later, we will introduce it here.

An order is an (R)-structure, where R is a 2-ary relation. We give some of the properties a 2-ary relation might have and more importantly the ones it has to possess to actually be an order. Usually R is denoted $<$ or $\leqslant$ in the context of orders.

Example 1.6 (orders):
Let X be a set and $R \subseteq X^2$ be a 2-ary relation on X.

- *The relation R is* reflexive, *if $(a,a) \in R$ holds for all $a \in X$.*
- *The relation R is* symmetric, *if $(a,b) \in R$ implies $(b,a) \in R$ for $a,b \in X$.*
- *The relation R is* antisymmetric, *if $(a,b) \in R$ and $(b,a) \in R$ implies $a = b$.*
- *The relation R is* transitive, *if $(a,b) \in R$ and $(b,c) \in R$ implies $(a,c) \in R$.*
- *The relation R is* total, *if for all $a,b \in X$ either $(a,b) \in R$ or $(b,a) \in R$.*

A partial order *on X is a 2-ary relation that is reflexive, antisymmetric and transitive. A* linear order *is a 2-ary relation that is antisymmetric, transitive and total. Linear orders are sometimes also called total orders. A* well-order *is a total order such that every non-empty subset of X has a least element.* □

1.3 Semigroups and Monoids

This section will deal with semigroups and monoids. The most important notions to be introduced will be homomorphisms, congruences and the free monoid. Building on this we will introduce monoid presentations. These will play an important role in the theory of automatic groups in later sections.

Semigroups and monoids are among the simplest algebraic structures. They only consist of a universe and a binary operation, usually called concatenation. Monoids could be described as a mathematical abstraction of a set of actions that can be sequenced. All the definitions in this section will be given for monoids, but most of them also hold for semigroups. Note,that there are a few differences between the theories of semigroups and monoids, which stem from the absence of an identity in a semigroup.

We define semigroups and monoids as follows.

Definition 1.7 (Semigroup and Monoid):
A semigroup *is an algebraic structure $\mathfrak{S} = (S, \cdot)$, where $\cdot : S \times S \to S$ is a binary associative operation, that is*

$$(a \cdot b) \cdot c = a \cdot (b \cdot c) \text{ for all } a, b, c \in S.$$

An element e of S is called an identity element *or* identity, *if*

$$e \cdot a = a \cdot e = a$$

holds for all a in S. An identity element is unique if it exists.
A semigroup that contains an identity element is called a monoid. □

Sometimes we want to add an order relation to a monoid or a semigroup. We do this by extending the monoid structure by an order relation thus having $\mathfrak{M} = (M, \cdot, <)$ where $<$ is an order on M. As usual we leave out the concatenation operator where it is possible.

Every semigroup can be made into a monoid by simply adjoining an identity and extending the concatenation operation appropriately.

We give a few examples of semigroups and monoids. The trivial semigroup is the empty set and the trivial monoid is the singleton set $\{a\}$ with $a \cdot a = a$.
Let $\mathfrak{N} = (\mathbb{N}, +)$ be the set of natural numbers including zero together with addition. This structure forms a semigroup and even a monoid where 0 is the identity element. Leaving out zero we get $\mathfrak{N}' = (\mathbb{N} \setminus \{0\}, +)$, which is a semigroup and a subsemigroup of $\mathfrak{N}$ but not a submonoid of $\mathfrak{N}$ as it does not contain an identity.
Let X be an arbitrary set and let $\mathcal{X}$ be the set of binary relations on X. We compose two binary relations R and S by defining $R \circ S := \{(a,b) \mid \text{There is a } c \text{ such that } (a,c) \in R \text{ and } (c,b) \in S\}$. Then, $(\mathcal{X}, \circ)$ is a semigroup and a monoid under the defined composition.
The set $X^X = \{f : X \to X\}$ of all maps from X to X forms a monoid together with composition of maps, where the identity element is the identity map. This monoid is a submonoid of $(\mathcal{X}, \circ)$.
For a slightly different example, let X be a totally ordered set and $\min\{a,b\}$ be the minimum of a and b. Then $(X, \min)$ is also a semigroup, and if X contains an an element e with $\min\{e,a\} = e$ for all $a \in X$, then $(X, \min)$ forms a monoid.

We usually want to relate different semigroups or monoids to each other.

Definition 1.8 (homomorphism):
Let $\mathfrak{S}$ and $\mathfrak{T}$ be two semigroups. A map $\varphi : S \to T$ is a semigroup homomorphism, *if $\varphi(a \cdot b) = \varphi(a)\varphi(b)$. If $\mathfrak{S}$ and $\mathfrak{T}$ are monoids and $\varphi(e_{\mathfrak{S}}) = e_{\mathfrak{T}}$, then φ is a* monoid homomorphism. □

We call injective homomorphisms monomorphisms, surjective homomorphisms epimorphisms and bijective homomorphisms isomorphisms.

Having introduced homomorphisms, the next notion that comes to mind are quotient structures. In the course of this thesis we will need to consider quotients of monoids. To accomplish this, we need to define equivalence relations that form a monoid under concatenation again.
Let $\mathfrak{M}$ be a monoid and $\sim$ be an equivalence relation on M. Then $\sim$ is called a *right congruence*, if $a \sim b$ implies $ac \sim bc$, it is called a *left congruence*, if $a \sim b$ implies $ca \sim cb$ and $\sim$ is called a *congruence* if both implications hold for all a, b, c in M. The congruence class of an element a is denoted by $[a]$.
Using a congruence we form a new monoid.

Lemma 1.9 (monoid modulo congruence):
Let $\mathfrak{M}$ be a monoid and $\sim$ be a congruence on $\mathfrak{M}$. Then the set of congruence classes $M/_{\sim} := \{[x] \mid x \in M\}$ together with concatenation defined by $[x] \cdot [y] := [xy]$ forms a monoid. In particular concatenation is well-defined and there is a unique identity element.

Proof: Let $x \sim x'$ and $y \sim y'$. We have to show that $xy \sim x'y'$. Because $\sim$ is a congruence, $xy \sim x'y$ holds. Analogously $x'y \sim x'y'$. In conclusion $xy \sim x'y \sim x'y'$.

Furthermore $[x][e_{\mathfrak{M}}] = [xe_{\mathfrak{M}}] = [x] = [e_{\mathfrak{M}}x] = [e_{\mathfrak{M}}][x]$. ■

The kernel of a homomorphism is a well established notion and we will define a kernel for monoid homomorphisms in the following way.

Remark 1.10 (kernel of monoid homomorphism):
Let $\mathfrak{M}$ and $\mathfrak{N}$ be monoids and $\varphi : \mathfrak{M} \to \mathfrak{N}$ be a monoid homomorphism.

Define the relation $\approx_\varphi$ by $a \approx_\varphi b$ if and only if $\varphi(a) = \varphi(b)$. This relation is a congruence on $\mathfrak{M}$, called the kernel of φ, *denoted* $\ker\varphi$.

Proof: It is clear that $\approx_\varphi$ is an equivalence relation on M. To prove that $\approx_\varphi$ is a congruence, let a, b, c be elements of M and $a \approx_\varphi b$. By the definition of a monoid homomorphism, $\varphi(ac)$ equals $\varphi(a)\varphi(c)$ which equals $\varphi(b)\varphi(c)$ by assumption and in conclusion this equals $\varphi(bc)$ which implies $ac \approx_\varphi bc$ by the definition of $\approx_\varphi$. The proof for $ca \approx_\varphi cb$ is analogous. ■

Homomorphism theorems are a fundamental concept in the examination of algebraic structures, homomorphisms and factor structures. Using the preceding paragraphs we can formulate a homomorphism theorem for monoids as follows.

Theorem 1.11 (homomorphism theorem for monoids):
Let $\mathfrak{M}$ and $\mathfrak{N}$ be monoids and $\varphi : \mathfrak{M} \to \mathfrak{N}$ be a monoid homomorphism. Then $\varphi = \sigma \circ \pi$ with $\pi(a) := [a]$ being the canonical homomorphism that maps every element of M to its congruence class under $\approx_\varphi$, and σ is an injective monoid homomorphism with $\sigma([a]) = \varphi(a)$.

In particular, $\operatorname{im}\varphi \cong \operatorname{im}\sigma$ *and if φ is surjective we have $\mathfrak{M}/_{\ker\varphi} \cong \mathfrak{N}$.*

The following diagram commutes.

$$\begin{tikzcd}
M \arrow[rr, "\varphi"] \arrow[dr, "\pi"'] & & N \\
& M/_{\approx_\varphi} \arrow[ur, "\sigma"'] &
\end{tikzcd}$$

Proof: We have to show, that σ is well-defined on the equivalence classes of $\approx_\varphi$ and injective.

To show, that σ is well-defined, let a and b be elements of M with $a \approx_\varphi b$. This means that $\varphi(a) = \varphi(b)$ which implies $\sigma([a]) = \sigma([b])$.

For injectivity let $\sigma([a]) = \sigma([b])$. Then $\varphi(a) = \varphi(b)$, thus $[a] = [b]$.

If φ is surjective, then σ is surjective, because $\varphi = \sigma \circ \pi$.

The notion of a free monoid captures the minimum of what is needed to form a monoid out of an arbitrary set X.

Definition 1.12 (free monoid):
A monoid $\mathfrak{F}$ is called free on a subset X of F *if it satisfies the universal property, that for any monoid $\mathfrak{M}$ and any map $f : X \to M$ there exists a unique monoid homomorphism $\varphi : F \to M$ that extends f, that is $f(a) = \varphi(a)$ for all $a \in X$.*

This can also be expressed using the following commutative diagram where $\iota : X \to \mathfrak{F}$ simply denotes the inclusion of X into $\mathfrak{F}$.

$$\begin{tikzcd}
X \arrow[r, "\iota"] \arrow[dr, "f"'] & \mathfrak{F} \arrow[d, "\exists^{=1}\varphi"] \\
 & \mathfrak{M}
\end{tikzcd}$$

□

We will give a constructive version of a free monoid on a finite set X in Definition 1.15 ensuring existence of a free monoid. We justify speaking of *the* free monoid on a set by showing that a free monoid on a set is unique up to isomorphism. Once we did that we use the notation $\mathcal{M}(X)$ for the free monoid on the set X.

Theorem 1.13 (free monoid is unique):
Let X and Y be sets and let $\mathfrak{M}$ be a monoid that is free on X and $\mathfrak{N}$ be a monoid that is free on Y. Then $\mathfrak{M}$ is isomorphic to $\mathfrak{N}$ if and only if there is a bijective map from X to Y.

Proof: Let X and Y be sets and $f : X \to Y$ be a bijective map.

The following diagram helps clearing up the situation.

$$\begin{tikzcd}
X \arrow[r, "\iota_X"] \arrow[d, "f"'] & \mathfrak{M} \arrow[d, shift right=2, "\exists^{=1}\varphi"'] \\
Y \arrow[r, "\iota_Y"'] & \mathfrak{N} \arrow[u, shift right=2, "\exists^{=1}\psi"']
\end{tikzcd}$$

A little exercise in diagram chase yields the result.

We first use the universal property on $\iota_Y \circ f$, which implies the existence of a unique homomorphism $\varphi : \mathfrak{M} \to \mathfrak{N}$ that extends $\iota_Y \circ f$. The other way around we use $\iota_X \circ f^{-1}$ and get $\psi : \mathfrak{N} \to \mathfrak{M}$.

We show that φ is an isomorphism and its inverse is ψ. We observe that

$$\begin{aligned}
\psi \circ \varphi \circ \iota_X &= \psi \circ \iota_Y \circ f \\
&= \iota_X \circ f^{-1} \circ f \\
&= \iota_X,
\end{aligned}$$

thus $\psi \circ \varphi$ extends ι_X. Because the identity on $\mathfrak{M}$ also extends ι_X and uniqueness, we have $\psi \circ \varphi = id_{\mathfrak{M}}$. We can do the same for $\varphi \circ \psi$, therefore $\varphi^{-1} = \psi$ and φ is the isomorphism between $\mathfrak{M}$ and $\mathfrak{N}$ we want.

In the following we only prove the theorem for finite sets X and Y, because otherwise we would have to deal with issues concerning infinite cardinalities. Assume that $\mathfrak{M}$ is free on a set X and $\mathfrak{N}$ is free on a set Y and that $\mathfrak{M}$ and $\mathfrak{N}$ are isomorphic. Let $\mathfrak{Z}$ be a monoid of cardinality 2. Then the homomorphisms from $\mathfrak{M}$ to $\mathfrak{Z}$ are in one to one correspondence to the maps from X to Z by the definition of a free monoid and the same holds for $\mathfrak{N}$ and Y. Thus there are $2^{|X|}$ homomorphisms from $\mathfrak{M}$ to $\mathfrak{Z}$ and $2^{|Y|}$ homomorphisms from $\mathfrak{N}$ to $\mathfrak{Z}$. Because $\mathfrak{N}$ and $\mathfrak{N}$ are isomorphic, we know that $2^{|X|} = 2^{|Y|}$ and thus $|X| = |Y|$, which ensures the existence of a bijective map $f : X \to Y$. ■

We now want to look at a way to specify congruences on monoids directly without the explicit use of a homomorphism. We do this by specifying a set of elements to be considered equal in the quotient monoid to be formed. We give a set of equations $v = w$ which we also denote by pairs (v, w). A congruence generated by a set of equations is the smallest congruence on a monoid that contains all of the generating equations.

Definition 1.14 (congruence generated by pairs):
Let $\mathfrak{M}$ be a monoid and let $R \subseteq M \times M$ be a set of pairs of elements of M. The congruence $\langle R \rangle$ generated by R is the smallest congruence on M containing all pairs in R. If $\mathcal{C}$ is the set of all congruences on M, then

$$\langle R \rangle := \bigcap_{\substack{C \in \mathcal{C} \\ R \subseteq C}} C.$$

□

Taking together the previous results enables us to establish the notion of a *monoid presentation*. A monoid presentation is a means to define a monoid $\mathfrak{M}$ by a quotient of the free monoid on a set. To show the existence of such a presentation for any monoid $\mathfrak{M}$, we use M as generating set. The free monoid $\mathcal{M}\ (M)$ exists by Theorem 1.16. We define a surjective monoid homomorphism φ from $\mathcal{M}\ (M)$ to $\mathfrak{M}$ by extending id_M to the unique monoid homomorphism $\varphi : \mathcal{M}\ (M) \to \mathfrak{M}$ that exists by Definition 1.12.

By the homomorphism theorem,

$$M \cong \mathcal{M}\ (M) \big/ \ker \varphi.$$

We take R to be a generating set for the kernel of φ and denote a presentation of $\mathfrak{M}$ as a quotient of the free monoid by

$$\mathrm{Mon}\langle\, M \mid R \,\rangle := \mathcal{M}\ (M) \big/ \langle R \rangle.$$

At a first glance this seems of no great value, as neither M nor $\ker \varphi$ need to be finite. However, there are many interesting cases in which we can choose a finite sets X and R even for infinite monoids. Monoids that allow for such a finite presentation are called *finitely presentable* and a monoid that is given by a finite presentation is *finiely presented*. A monoid can have infinitely many presentations. In particular, there are infinitely many presentations for the trivial monoid.

It is in general undecidable whether two monoid presentations present isomorphic monoids. In particular, we cannot in general decide whether a given finitely presented monoid is trivial.

We will now construct a free monoid from a given finite set A. We assume A to be finite, because this is the case of interest in this thesis. Note that this construction also works for infinite sets. Because we will talk about formal languages in the next section, we call A an alphabet, thus an alphabet is nothing else than a finite set.

After giving the construction we show that the constructed monoid is in fact the free monoid on A.

We start with a few straightforward constructions based on the notion of an alphabet.

Definition 1.15 (constructive free monoid):
Let A be a finite set. We call A an alphabet and if we give an additional ordering relation $<$ on A we call $(A, <)$ an ordered alphabet.

- *Let n be a natural number. A* string *s of length n over A is an n-tuple $s \in A^n$. We denote the length of s by $|s|$.*

- *For $n = 0$ there is one unique string called the* empty string *or* nullstring *also denoted ε_A or ε if no ambiguity arises.*
- *The set of all strings over A is denoted A^* and is defined as the union of A^n for all $n \in \mathbb{N}$, the set of all non-empty strings over A is denoted A^+.*
- *The operation*

$$\cdot : A^* \times A^* \to A^* \,:\, ((s_1, \ldots, s_n), (t_1, \ldots, t_m)) \mapsto (s_1, \ldots, s_n, t_1, \ldots, t_m)$$

  is called concatenation *of the strings s and t.* □

It is immediate, that A^* together with concatenation forms a monoid. To conclude this section we show that the constructed monoid is the free monoid on A. We will come back to the theory of monoids in later sections.

Theorem 1.16 (existence of a free monoid):
Let A be a finite set. The structure $\mathfrak{M} = (A^*, \cdot)$, the set of all strings over A together with the concatenation operation forms a monoid and is isomorphic to $\mathcal{M}(A)$.

Proof: It is clear that $\mathfrak{M}$ is a monoid. To show that $\mathfrak{M}$ is free on A all that has to be done is to check, whether the universal property 1.12 holds for $\mathfrak{M}$.

Let $\mathfrak{N}$ be a monoid and $f : A \to \mathfrak{N}$ be a map. We have to show the existence of a unique monoid homomorphism $\varphi : \mathfrak{M} \to \mathfrak{N}$. Existence is easy to show. For an element $v = v_1 v_2 \cdots v_n$ of $\mathfrak{M}$ we extend f to a monoid homomorphism φ by setting $\varphi(v_1 v_2 \cdots v_n) := f(v_1) f(v_2) \cdots f(v_n)$. For uniqueness suppose there is another monoid homomorphism φ' extending f. This would mean that $\varphi' = \varphi$, and thus φ is unique. ■

1.4 Regular Languages and Finite State Automata

Regular languages, sometimes also called rational languages, have been studied extensively in the last fifty years. The reason behind this is twofold. On the one hand, the concept of regular languages captures simple discrete processes with finite state, and thus is of great practical interest. Also, regular languages have a particularly nice theory: interesting computational problems are not only decidable, but also in many cases, efficiently so. Resulting from this there is a very rich body of research on regular languages.

As already established in the previous section, alphabets will be denoted by uppercase latin letters like A or B and are nothing else than finite sets. Examples of alphabets are $\{0, 1\}$ or $\{x, y\}$, but also all symbols on a keyboard, the greek alphabet or a set of dominos form an alphabet.

Languages are sets of strings, they are allowed to be empty, finite or infinite.

Definition 1.17 (language):
Let A be an alphabet. A language *L over A is a subset of A^*.* □

Although languages are subsets of the free monoid on an alphabet, they are not submonoids in most cases.

For $s, x, y, z \in A^*$ and $s = xyz$, where any of x, y and z may be the empty string, we call x a *prefix* of s and denote it by $x \preceq s$, we call y an *infix* of s, denoted by $y \curlyvee s$ and z a *suffix* of s denoted by $z \succeq s$. If we are given a string s and want to look at prefixes of a given length $t \in \mathbb{N}$ we write $s[t]$ for the prefix of length t of s. The prefix of length zero is the empty string ε and if t exceeds $|s|$ then $s[t]$ equals s.

We now turn our attention to the question of how to define a subset of A^*. As it turns out there are many possible ways to accomplish this task and not all of these are equivalently powerful in the sense that there are subsets of A^* that cannot be defined by certain methods. We can define subsets of A^* by set-theoretic means, logical formulae, regular expressions, grammars or, as we will see in the following by finite state automata or semigroups.

There are subsets of A^* that do not allow for a Turing machine that computes their characteristic function. The existence of such sets is proven very easily by a purely combinatoric argument. There are only countably many Turing machines but the powerset of A^* is uncountable. Thus we cannot hope to find for all subsets of A^* a Turing machine that decides whether a string over A is an element of the subset.

The concept of a finite state automaton is a simple and clean concept that has been well established in the theory of computation and the theory of discrete systems. Systems with finite state are found in many applications in computer science, but we will introduce a very relevant application in group theory later on.

Definition 1.18 (finite state automaton):
A finite state automaton $\mathfrak{A}$ is a quintuple (Q, A, I, F, τ) consisting of

- *A finite set of states Q.*
- *A finite alphabet A.*
- *A set $I \subset Q$ of start states.*
- *A set $F \subset Q$ of final states.*
- *A transition relation τ, which is a subset of $Q \times (A \cup \{\varepsilon\}) \times Q$.* □

This structure can be seen as a directed labelled graph with set of vertices Q and edges (q_1, a, q_2) between states q_1 and q_2 with label a for each (q_1, a, q_2) in τ. The sets S and F are predicates of states. An automaton $\mathfrak{A}$ is *deterministic* if there is only one initial state, usually denoted by q_0 and τ is a total map $\tau : Q \times A \to Q$.

A *run* on $\mathfrak{A}$ is a finite sequence $\mu = q_0 a_1 q_1 a_2 \cdots a_{n-1} q_n$, such that q_0 is an element of I and there exist transitions $\tau_i = (q_{i-1}, a_i, q_i)$ in τ for $1 \leqslant i \leqslant n$. A run is called *accepting* if additionally q_n is an element of F. A *partial run* is a run that starts in an arbitrary state. The observant reader might have noticed, that reading off the labelling of a run μ yields a string $s = a_1 a_2 \cdots a_n$ over the alphabet A.
Given a deterministic automaton and a string s of length n over A, we define a unique run $\mu(s)$ by

$$\mu(s) := q_0 s_1 q_1 s_2 q_1 \cdots q_{n-1} s_n q_n,$$

where q_0 is the unique state in I and q_{i+1} is uniquely determined by the total map τ. The automaton is said to *accept* the string s, if $\mu(s)$ is an accepting run. For non-deterministic automata there might be none or more than one run for a string s. A non-deterministic automaton accepts a string s if there exists an accepting run of the automaton on s.

The following definition shows how the concept of a finite state automaton is relevant in the theory of formal languages. A finite state automaton defines a subset $L(\mathfrak{A})$ of A^*.

Definition 1.19 (language of finite state automaton):
Let $\mathfrak{A} = (Q, A, I, F, \tau)$ be a finite state automaton over an alphabet A. The language of $\mathfrak{A}$, *denoted by $L(\mathfrak{A})$, is defined as the set of all strings accepted by $\mathfrak{A}$.*

$$L(\mathfrak{A}) := \{w \mid w \in A^* \ \ \mathfrak{A} \text{ accepts } w\} \qquad \square$$

Having established the notion of the language of a finite state automaton we can now define regular languages.

Definition 1.20 (regular language):
We call a subset $L \subseteq A^*$ of the strings over a finite alphabet A recognizeable *or* acceptable *by a finite state automaton or* regular, *if there is a finite state automaton $\mathfrak{A}$ over A with $L(\mathfrak{A}) = L$.* $\square$

In standard literature regular languages are usually defined by regular expressions. We do not want to go through the trouble of defining regular languages by regular expressions and then show equivalence of the definable languages. We refer the reader to standard literature on formal language theory such as [HMU06].

For every regular language L there is a uniquely determined deterministic finite state automaton $\mathfrak{A}$, which has minimal state count among all automata with language L. As usual uniquely determined means unique up to isomorphism of finite state automata. This is proven constructively in the following, although some of the proofs are sketchy.

Let L be a regular language. From the definition of a regular language we know that there is a finite state automaton $\mathfrak{A}$ with $L(\mathfrak{A}) = L$. First we construct a deterministic automaton $\mathfrak{A}'$ with the same language. After that we construct a third finite state automaton $\mathfrak{A}''$ that has minimal state count among all deterministic finite state automata accepting L.

The procedure to construct a deterministic finite state automaton from an arbitrary finite state automaton is called powerset construction.

We give a formal definition of powerset construction but leave the full proof of correctness to the reader. The important points to show are certainly that $L(\mathfrak{A}) = L(\mathcal{P}(\mathfrak{A}))$ and that the transition relation is in fact a well-defined total map on the states of $\mathcal{P}(\mathfrak{A})$.

The powerset construction introduced here blows up the number of states exponentially. This does not have to be the case so we can be smart and build a powerset automaton stepwise. A practical implementation would form the state sets during the computation. This does, however, not prevent the powerset automaton from having exponentially many states in the worst case. The interested reader might think of a very easy to find example where this is the case as an exercise.

As preparation for the powerset construction we define the epsilon closure $C_\varepsilon(S)$ of a set S of states. Informally this is the set of states that are reachable from states inside S by epsilon transitions only.

Definition 1.21 (epsilon closure):
Let $\mathfrak{A} = (Q, A, I, F, \tau)$ be a finite state automaton and $S \subseteq Q$ be a set of states. Define

$$C_\varepsilon^0 := S,$$

and

$$C_\varepsilon^{n+1} := \{q \mid (p, \varepsilon, q) \in \tau \text{ for } p \in C_\varepsilon^n\}.$$

The epsilon closure $C_\varepsilon(S)$ of S is now defined as

$$C_\varepsilon := \bigcup_{n \in \mathbb{N}} C_\varepsilon^n \qquad \square$$

We now define the powerset automaton.

Definition 1.22 (powerset construction):
Let $\mathfrak{A} = (Q, A, I, F, \tau)$ be an arbitrary finite state automaton. The powerset automaton $\mathcal{P}(\mathfrak{A})$ is defined as follows.

$$\mathcal{P}(\mathfrak{A}) := (\mathcal{P}(Q), A, C_\varepsilon(I), F_p, \delta),$$

where S is an element of F_p if $F \cap S$ is not empty. We define δ as a map from $\mathcal{P}(Q) \times A$ to $\mathcal{P}(Q)$ in the following way.

$$\delta : \mathcal{P}(Q) \times A \to \mathcal{P}(Q) \,:\, (S, a) \mapsto C_\varepsilon(\{q \mid (p, a, q) \in \tau,\ p \in S,\ a \in A \cup \{\varepsilon\}\}) \qquad \square$$

We now address state count. We show that every regular language defines a unique minimal deterministic finite state automaton up to isomorphism. Given a finite state automaton $\mathfrak{A}$ for a regular language L, this is done by defining an equivalence relation on the states of $\mathfrak{A}$. The resulting quotient automaton is the unique minimal automaton for L, which is shown by giving a surjective homomorphism of finite state automata onto the quotient automaton.

Let $\mathfrak{A}$ be a finite state automaton with language L. For a state q of $\mathfrak{A}$, we define the language accepted from q to be the set of strings, for which a partial accepting run from q exists. This language is denoted $L(q)$.

Definition 1.23 (state equivalence):
Let $\mathfrak{A} = (Q, A, \{q_0\}, F, \tau)$ be a deterministic finite state automaton. Define an equivalence relation on the state set Q by

$$p \sim q :\Leftrightarrow L(p) = L(q)\,.$$

The quotient automaton $\mathfrak{A}/_\sim$ consists of the following components.

- *The set of states $Q/_\sim = \{[q] \mid q \in Q\}$.*
- *The alphabet A.*
- *The initial state $[q_0]$.*
- *The set of accept states $F/_\sim = \{[q] \mid q \in F\}$.*
- *The transition function $\tau/_\sim([q], a) = [\tau(q, a)]$.* $\square$

For well-definedness of $\tau/_\sim$, note that $p \sim q$ implies $L(p) = L(q)$ which implies $L(\tau(p, a)) = L(\tau(q, a))$ for all a in the alphabet A. Thus $p \sim q$ implies $\tau/_\sim([p], a) = \tau/_\sim([q], a)$.

The Nerode congruence of a language L defines congruence classes of strings in A^* and will later be used to define a finite state acceptor.

Definition 1.24 (Nerode congruence):
Let L be a language over an alphabet A. Then we define a right congruence $\equiv_L$ on A^* by

$$x \equiv_L y \;:\Longleftrightarrow\; xv \in L \Leftrightarrow yv \in L \text{ for all } v \in A^*.$$

This congruence is called the Nerode congruence of L. $\square$

A language with only finitely many equivalence classes under the Nerode congruence yields a natural way to form a finite state automaton.

Definition 1.25 (Nerode automaton):
Let L be a language over an alphabet A such that $\equiv_L$ has finitely many equivalence classes. We define the deterministic finite state automaton $\mathfrak{A} = (Q, A, I, F, \tau)$ consisting of

- *Q is the set $\{[s]_{\equiv_L} \mid s \in A^*\}$ of congruence classes of $\equiv_L$.*
- *I is $\{[\varepsilon]\}$, the equivalence class of the empty string.*
- *$[s]_{\equiv_L}$ is an element of F if and only if s is an element of L.*
- *$\tau([s]_{\equiv_L}, a) = [sa]_{\equiv_L}$.* □

The following theorem implies that the automaton defined in L accepts precisely L and that the automaton is a unique minimal deterministic finite state automaton for the given regular language.

Theorem 1.26 (Myhill-Nerode):
Let L be a language over an alphabet A. Then L is regular if and only if there are only finitely many congruence classes with respect to $\equiv_L$.

Proof: Let L be a language that has finitely many congruence classes under $\equiv_L$. Let $v = v_1 v_2 \dots v_n$ be a string over A. Then the run of the Nerode automaton ends in the state $[v]$, which is contained in F if and only if v is an element of L. We note that either all strings in a congruence class are in L or none is, because if the converse was the case we would immedeately have a contradiction to the definition of the Nerode congruence. Thus the automaton given in Definition 1.25 accepts L.

Conversely let L be a regular language accepted by a deterministic finite state automaton $\mathfrak{A}$, which we assume to have no states that are not reachable from q_0. We form equivalence classes of states of $\mathfrak{A}$ that correspond to equivalence classes of $\equiv_L$. Take the equivalence relation given in Definition 1.23 on the states of $\mathfrak{A}$. All strings t with $\tau(q_0, t)$ in the same equivalence class of states are also equivalent under the Nerode congruence, thus we map each equivalence class $[q]$ of states to $[t]_{\equiv_L}$ for a t in A^*, such that $\tau(q_0, t)$ is an element of $[q]$. Such a t exists because we assumed that each state is reachable from q_0. This map is well defined and it is surjective. For surjectivity we observe that a class $[v]$ has the state of $\mathfrak{A}$ mapped to it that is reached by the run of $\mathfrak{A}$ on v. Thus there only are finitely many equivalence classes under the Nerode congruence. ■

As a corollary we find that the Nerode automaton is the unique minimal finite state automaton for L. Construction of this automaton involves finding equivalence classes of states, which is done by seperating classes of states iteratively.

Corollary 1.27 (minimal automaton):
Let L be a regular language. The Nerode automaton is the unique deterministic finite state automaton with language L up to isomorphism with minimal state count.

Proof: Let $\mathfrak{A}$ be a deterministic finite state automaton for L and $\mathfrak{N}$ be the Nerode automaton for L. Then there is a surjective homomorphism of finite state automata from $\mathfrak{A}$ to $\mathfrak{N}$ by the proof of 1.26. In particular, $\mathfrak{N}$ has less than or equally many states as $\mathfrak{A}$. ■

Minimisation can be done efficiently in $O(|Q| \log |Q|)$ where $|Q|$ is the number of states. In conclusion, deterministic and non-deterministic finite state automata have the same expressive power. They both can define regular languages. Powerset construction and minimisation are well known and we refer to standard literature for further treatment. We note that non-deterministic finite state automata do in general not have a unique smallest equivalent. A non-deterministic automaton for a regular language can potentially be exponentially smaller than a deterministic one. Minimal non-deterministic finite state automata are not unique and finding them is a very hard algorithmic problem: NFA-MIN is PSPACE-hard.

Finite state automata are an appropriate tool to look at regular languages algorithmically and there is a very complete collection of algorithms to deal with finite state automata. However, finite state automata do not allow to look at algebraic properties a regular language might possess. Recognition of subsets of A^* by finite semigroups has turned out to be a better tool for this purpose. An introduction to the theory of syntactic semigroups can be found in [Pin00].

Definition 1.28 (recognition by semigroup):
Let $\mathfrak{S}$ be a semigroup and $\mathfrak{T}$ be a finite semigroup. A subset $L \subseteq S$ of S is recogniseable by *$\mathfrak{T}$, if there exists a surjective semigroup homomorphism φ from $\mathfrak{S}$ to $\mathfrak{T}$ and a subset $F \subseteq T$ of T such that $L = \varphi^{-1}(F)$.* □

We inspect the connection between the concept of languages defineable by finite state automata and the ones defineable by semigroup recognition. Our aim is to show, that the languages that are recognizeable by a finite semigroup are exactly the ones that are acceptable by finite state automata.

The idea behind the proof is that for a deterministic finite state automaton $\mathfrak{A}$ we can define τ in a slightly different way, namely by a map $\mu : A \to \mathrm{End}(Q)$ where $\mathrm{End}(Q)$ is the set of all maps $f : Q \to Q$. In this way, we define a state transformation for every symbol in the alphabet. We can extend μ to a monoid homomorphism μ^* from A^* to $\mathrm{End}(Q)$. A string s is accepted by the automaton $\mathfrak{A}$, if and only if $\mu^*(s)(I)$ contains an element in F.

In the opposite direction, given a finite semigroup accepting a subset of a semigroup we can construct a finite state automaton. Making this idea a bit more precise, if we are given a semigroup homomorphism $\varphi : A^* \to T$ and a subset F of T, we define a finite state automaton $\mathfrak{A} = (T, A, \{e\}, F, \tau)$ where $\tau = \{(x, a, x \circ \varphi(a)) \mid x \in T, a \in A\}$.

Using these ideas we prove the following theorem.

Theorem 1.29 (recogniseable iff acceptable):
A subset L of A^* is recognizeable by a finite semigroup if and only if there is a finite state automaton for L.

Proof: Assume that L is a subset of A^* that is accepted by a deterministic finite state automaton $\mathfrak{A} = (Q, A, \{q_0\}, F, \tau)$. The semigroup of endomaps $\mathrm{End}(Q)$ is finite and the set

$$F' := \{f \in \mathrm{End}(Q) \mid f(q) \in F \text{ for } q \in I\}$$

is certainly a subset of $\mathrm{End}(Q)$. We define

$$\mu : A \to \mathrm{End}(Q) : a \mapsto (q \mapsto \tau(q, a)).$$

By the definition of the free semigroup on A we extend μ to a semigroup homomorphism from A^* to $\mathrm{End}(Q)$. It remains to show that s is accepted by $\mathfrak{A}$ if and only if $\mu(s)$ is an element of F'. Let s be a string over A. It is accepted by $\mathfrak{A}$ if and only if the run of $\mathfrak{A}$ on s starting at q_0 ends in an accept state of $\mathfrak{A}$. But this is precisely the case if and only if the state transformation induced by s takes q_0 to an accept state.

Conversely let L be a subset of A^* and $\mathfrak{T}$ be a finite monoid that recognizes L, that is there is a homomorphism φ from A^* to $\mathfrak{T}$ and a finite subset F of T such that $L = \varphi^{-1}(T)$. We construct a finite state automaton from this information. Let $\mathfrak{A} = (T, A, \{e\}, F, \tau)$ where $\tau(t,a) = t \circ \varphi(a)$. We need to show that a string s is accepted by $\mathfrak{A}$ if and only if it is recognised by $\mathfrak{T}$. Let s be an element of L, thus s is an element of $\varphi^{-1}(F)$ and $e \circ \varphi(s)$ is an element of F. Thus $\mathfrak{A}$ accepts s if and only if $\varphi(s)$ is an element of F.

■

In the following paragraph we want to examine closure properties of regular languages under a few set theoretic operations, such as map, intersection, union, concatenation, Kleene star and complement. As languages over an alphabet are subsets of A^*, all set operations also apply to languages.

Given two alphabets A and B and maps $f : A \to B$ and $g : A \to B^*$, these extend to maps $f : A^* \to B^*$ and $g : A^* \to B^*$. The effect of f is just replacing symbols in A by symbols in B, the map g replaces symbols in A by finite strings over B.

Given a language L, we look at subsets of L and the complement $\overline{L}$ of L in A^*. Given languages L_1 and L_2 over an alphabet A, the union $L_1 \cup L_2$ and the intersection $L_1 \cap L_2$ of L_1 and L_2 are also languages. The concatenation of L_1 and L_2 is defined as

$$L_1 \cdot L_2 := \{v \cdot w \mid v \in L_1, w \in L_2\}.$$

The set of all languages over an alphabet forms a monoid under concatenation of languages. The cyclic submonoid generated by a fixed language L is called the *Kleene closure* of L. We define finite powers of L as follows.

$$\begin{array}{rcl} L^0 & := & \{\varepsilon\} \\ L^n & := & L^{n-1} \cdot L \end{array}$$

The *Kleene closure* or *Kleene star* of L is then defined as the union of all finite powers of a language L as follows.

$$L^* := \bigcup_{n \in \mathbb{N}} L^n.$$

For an alphabet A, the Kleene closure is the set A^*.

Given two languages L_1 and L_2, possibly over distinct alphabets A_1 and A_2, we might define a direct product $L_1 \times L_2$ of L_1 and L_2 as the cartesian product of the two sets as a subset of ${A_1}^* \times {A_2}^*$. This would yield a set of pairs of strings, which is not quite satisfying because we do not have an alphabet in the strict sense over which the strings are formed. For pairs of strings to be accepted by a deterministic finite state automaton, we need the strings in a pair to be of the same length. This is because the length of a run of a deterministic automaton is well defined by the length of the string to be accepted. Thus we need to add a padding character to fill up shorter strings.

We solve these technical problems by introducing the notion of convolution of strings, which supplies us with a well defined alphabet and adds padding implicitly.

Definition 1.30 (convolution):
Let A be an alphabet, $\square \notin A$ a blank symbol and $s_1, s_2, \ldots, s_n$ strings over A. The convolution $s_1 \otimes s_2 \otimes \ldots \otimes s_n$ *of $s_1, \ldots, s_n$ is defined as a string of length*

$$l := \max\{|s_1|, |s_2|, \ldots, |s_n|\}$$

over $(A \cup \{\square\})^n$ in the following way

$$s_1 \otimes s_2 \otimes \cdots \otimes s_n := \begin{bmatrix} s'_{1,1} \\ s'_{2,1} \\ \vdots \\ s'_{n,1} \end{bmatrix} \begin{bmatrix} s'_{1,2} \\ s'_{2,2} \\ \vdots \\ s'_{n,2} \end{bmatrix} \cdots \begin{bmatrix} s'_{1,l} \\ s'_{2,l} \\ \vdots \\ s'_{n,l} \end{bmatrix}$$

where

$$s'_{i,j} = \begin{cases} s_{i,j} & \text{if } j \leqslant |s_i| \\ \square & otherwise \end{cases}.$$

and $s_{i,j}$ denotes the j-th character of s_i.

If L_1 and L_2 are languages, the convolution *of L_1 and L_2, denoted $L_1 \otimes L_2$ is defined as the set of convolutions of strings in L_1 and L_2*

$$L_1 \otimes L_2 := \{v \otimes w \mid v \in L_1, w \in L_2\}.$$

For a language L we denote the n-fold convolution of L with itself by $L^{\otimes n}$ and define it as

$$\begin{aligned} L^{\otimes 1} &:= L \\ L^{\otimes n+1} &:= L \otimes L^{\otimes n}. \end{aligned}$$

We denote the padded alphabet by $A^{\otimes n}$ which is nothing else than $A^n \setminus \{\square^n\}$. We denote an element of $A^{\otimes n}$ by

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$$

where a_i are elements of $A \cup \{\square\}$ and at least one of the a_i is not equal to $\square$. □

In a later application we permute components of a convolution of languages, so we define what we mean by this.

Remark 1.31 (permutation of strings):
Let A be an alphabet. The symmetric group $\mathfrak{S}_n$ acts on $A^{\otimes n *}$ via

$$. : \mathfrak{S}_n \times \left(A^{\otimes n}\right)^* \to \left(A^{\otimes n}\right)^* : (\sigma, s_1 \otimes \cdots \otimes s_n) \mapsto s_{\sigma(1)} \otimes \cdots \otimes s_{\sigma(n)}.$$

This action extends to an action on the languages over A, that is an action on $\mathcal{P}\left((A^{\otimes n})^*\right)$. □

Sometimes we want to look at all prefixes of strings in a language.

Definition 1.32 (prefix closure):
Let L be a language. Then the language $L_{\preceq}$ of all prefixes of strings in L is called the prefix closure *of L. If L is equal to $L_{\preceq}$, then L is called* prefix-closed. □

Given the overview of operations on languages, we are now interested in closure properties of regular languages under these operations. Closure under an operation means that, given regular languages as input, the operation yields a regular language as a result.

To prove closedness under an operation we just need to ensure the existence of a finite state automaton that accepts the resulting language. We start with a negative answer. In this case we have to show that there cannot be any finite state automaton for the language in question.

Lemma 1.33 (subsets not regular):
The language A^* for an alphabet A is regular. There are subsets L of A^* that are not regular. In particular the class of regular languages is not *closed under taking subsets.*

Proof: The language A^* is accepted by a finite state automaton with only one state q, which is initial and final and for all a in A there is a transition (q,a,q). The powerset $\mathcal{P}(A^*)$ is uncountable, yet there are only countably many finite state automata. Thus there have to be subsets of A^* that are not decidable by a finite state automaton. Using the Nerode congruence, we can prove that the language

$$L := \{a^n b^n \mid n \in \mathbb{N}\}$$

is not regular. But it certainly is a subset of $\{a,b\}^*$. ■

We now go on with a series of positive answers. If we talk about sets, we almost certainly will have to talk about maps. Regular languages are closed under maps between alphabets.

Lemma 1.34 (map preserves regularity):
Let A and B be alphabets and L_A be a regular language over A. Let $f : A \to B$ and $g : A \to B^*$ be maps. We denote by f and g also the extensions of f and g to maps from A^* to B^*. The images $f(L_A)$ and $g(L_A)$ are regular languages over B. Additionally, if L_B is a regular language over B, the the inverse image $f^{-1}(L_B)$ is a regular language over A.

Proof: Let $\mathfrak{A}_A$ and $\mathfrak{A}_B$ be finite state automata that accept L_A and L_B respectively.

We give a finite state automaton for $f(L_A)$ by

$$\mathfrak{A}'_A := (Q,B,I,F,\tau')$$

consisting of the state sets Q, I and F of $\mathfrak{A}_A$ and the transition relation τ', that contains a transition $(p,f(a),q)$, for each transition (p,a,q) in the transition relation of $\mathfrak{A}_A$. If a string s is accepted by $\mathfrak{A}_A$, then there is a run $\mu = q_0 s_1 q_1 \cdots q_{n-1} s_n q_n$ of $\mathfrak{A}_A$ on s. By construction, $\mu' = q_0 f(s_1) q_1 \cdots q_{n-1} f(s_n) q_n$ is an accepting run for $f(s)$.

For g we interpret $g(A)$ as an alphabet and map $\mathfrak{A}_A$ in the same way as for f. Certainly $g(L_A)$ is a subset of B^* and accepted by the image of $\mathfrak{A}_A$ under g.

The construction for the preimage $f^{-1}(L_B)$ is similar.

■

The next few lemmas deal with complement, union and intersection of regular languages. Regular languages are closed under all of these operations. This will enable us to interpret logical formulae with regular lanugages in a later paragraph.

Lemma 1.35 (complement):
Let L be a regular language over an alphabet A.Then $\overline{L} = A^* \setminus L$ is also a regular language.

Proof: Let $\mathfrak{A} = (Q, A, \{q_0\}, F, \tau)$ be a deterministic finite state automaton with $L(\mathfrak{A}) = L$. Then $\overline{\mathfrak{A}} = (Q, A, \{q_0\}, Q \backslash F, \tau)$ accepts a string s in A^* if and only if $\mathfrak{A}$ does not accept it. The claim is thus proven, but note that $\mathfrak{A}$ being deterministic is essential for this proof. ■

The direct product of two automata over an alphabet A accepts the intersection of two languages over that alphabet. We first give the definiton of a product automaton, and then give a proof that it is an automaton for the intersection of the languages of the two automata.

Definition 1.36 (product automaton):
Let $\mathfrak{A}_1 = (Q_1, A, \{q_{1,0}\}, F_1, \tau_1)$ and $\mathfrak{A}_2 = (Q_2, A, \{q_{2,0}\}, F_2, \tau_2)$ be deterministic finite state automata. We define the direct product $\mathfrak{A}_1 \times \mathfrak{A}_2$ of $\mathfrak{A}_1$ and $\mathfrak{A}_2$ as

$$\mathfrak{A}_1 \times \mathfrak{A}_2 := (Q, A, I, F, \tau)$$

consisting of

- *The state set $Q = Q_1 \times Q_2$.*
- *The alphabet A.*
- *The set I equal to $\{(q_{1,0}, q_{2,0})\}$.*
- *The set of final states $F := F_1 \times F_2$.*
- *The transition function*

$$\tau : (Q_1 \times Q_2) \times A \to Q_1 \times Q_2 : ((p,q), a) \mapsto (\tau_1(p,a), \tau_2(q,a)) \qquad \square$$

Lemma 1.37 (automaton for intersection):
Let $\mathfrak{A}_1$ and $\mathfrak{A}_2$ be deterministic finite state automata. The language that is accepted by the product automaton $\mathfrak{A}_1 \times \mathfrak{A}_2$ is the intersection $L(\mathfrak{A}_1) \cap L(\mathfrak{A}_2)$ of the languages accepted by $\mathfrak{A}_1$ and $\mathfrak{A}_2$.

Proof: Let s be a string over A, that is accepted by $\mathfrak{A}_1$ as well as $\mathfrak{A}_2$. This is, by definition, the case if and only if there are accepting runs

$$\mu_1(s) = q_{1,0} s_1 q_1^{(1)} \dots q_1^{(n-1)} s_n q_1^{(n)}$$

of $\mathfrak{A}_1$ on s and

$$\mu_2(s) = q_{2,0} s_1 q_2^{(1)} \dots q_2^{(n-1)} s_n q_2^{(n)}$$

μ_2 of $\mathfrak{A}_2$ on s. We give the accepting run μ on $\mathfrak{A}_1 \times \mathfrak{A}_2$ on s as follows.

$$\mu = (q_{1,0}, q_{2,0}) s_1 \left(q_1^{(1)}, q_2^{(1)}\right) s_2 \cdots s_n \left(q_1^{(n)}, q_2^{(n)}\right)$$

Thus the claim is proven. ■

A corollary of the above construction is that regular languages are closed under intersection.

Corollary 1.38 (intersection):
Let L_1 and L_2 be regular languages. Then the intersection $L_1 \cap L_2$ is also a regular language.

Proof: As L_1 and L_2 are regular, there are deterministic finite state automata $\mathfrak{A}_1$ and $\mathfrak{A}_2$ with $L(\mathfrak{A}_1) = L_1$ and $L(\mathfrak{A}_2) = L_2$. Thus by Lemma 1.37 there is a deterministic finite state automaton $\mathfrak{A}$ that accepts $L_1 \cap L_2$.

The union of two languages over an alphabet A is equal to the complement of the intersection of the complements of the respective languages, thus the union of regular languages is also regular. Note, that if we wanted to construct an automaton for the union of two regular languages directly, we would only have to adjust the set of accept states of the product automaton.

Corollary 1.39 (union):
Let L_1 and L_2 be regular languages over an alphabet A. Then the union $L_1 \cup L_2$ is also a regular language.

Proof: Obviously $L_1 \cup L_2 = \overline{\overline{L_1} \cap \overline{L_2}}$. This language is regular by Lemma 1.35 and Corollary 1.38. ■

Concatenation and Kleene star of regular languages are also regular. We use ε transitions, that need no input symbol for this construction. Note that these transitions were included in the definition of an automaton, and that the inclusion of ε transitions does not make automata more powerful.

Definition 1.40 (direct sum automaton):
Let $\mathfrak{A}_1 = (Q_1, A, \{q_{1,0}\}, F_1, \tau_1)$ and $\mathfrak{A}_2 = (Q_2, A, \{q_{2,0}\}, F_2, \tau_2)$ be deterministic finite state automata. We define the direct sum $\mathfrak{A}_1 \oplus \mathfrak{A}_2$ *of $\mathfrak{A}_1$ and $\mathfrak{A}_2$ as*

$$\mathfrak{A}_1 \oplus \mathfrak{A}_2 := (Q, A, I, F, \tau)$$

consisting of

- *The state set $q = Q_1 \dot{\cup} Q_2$.*
- *The alphabet A.*
- *The set I equal to $\{q_{1,0}\}$.*
- *The set F_2 of final states.*
- *The transition relation τ equal to $\tau_1 \dot{\cup} \tau_2$ plus ε transitions $(p, \varepsilon, q_{2,0})$ for all $p \in F_1$.* □

The direct sum of two automata accepts the concatenation of the languages of the two automata.

Lemma 1.41 (concatenation):
Let $\mathfrak{A}_1$ and $\mathfrak{A}_2$ be deterministic fintie state automata. Then the direct sum $\mathfrak{A}_1 \oplus \mathfrak{A}_2$ accepts the language $L(\mathfrak{A}_1) \cdot L(\mathfrak{A}_2)$.

Proof: Let $v \cdot w$ be a string in $L(\mathfrak{A}_1) \cdot L(\mathfrak{A}_2)$. Then $\mathfrak{A}_1 \oplus \mathfrak{A}_2$ accepts $v \cdot w$, because there are accepting runs $\mu(v)$ of $\mathfrak{A}_1$ on v and and $\mu(w)$ of $\mathfrak{A}_2$ on w. The concatenation of these runs is an accepting run for $v \cdot w$ on $\mathfrak{A}_1 \oplus \mathfrak{A}_2$. The concatenation is formed by inserting the transition $\left(q_1^{(n)}, \varepsilon, q_{2,0}\right)$ in between $\mu(v)$ and $\mu(w)$, which exists by construction.

If a string s is accepted by $\mathfrak{A}_1 \oplus \mathfrak{A}_2$, there is an accepting run μ of $\mathfrak{A}$ on s. This run has to pass through an accepting state of $\mathfrak{A}_1$ by definiton. We decompose s into $v \cdot w$ such that v is exactly the string that is read up to the accept state of $\mathfrak{A}_1$ and w is just the rest of s. Thus v is an element of $L(\mathfrak{A}_1)$ and w is an element of $L(\mathfrak{A}_2)$. ■

As a corollary again, the concatenation of regular lanugages is also regular.

Corollary 1.42 (concatenation regular):
Let L_1 and L_2 be regular language. Then $L_1 \cdot L_2$ is regular.

Proof: This follows directly from the preceding lemma. ■

Regularity is also closed under the Kleene star.

Lemma 1.43 (Kleene star):
Let L be a regular language. Then L^* is also regular.

Proof: Let $\mathfrak{A}$ be a deterministic finite state automaton that accepts L. We add a new state q_{accept} to the automaton and replace the set of accept states by a singleton set containing this new state q_{accept}. We add ε-transitions from all accept states of $\mathfrak{A}$ to q_0 and to q_{accept} and additionally an ε-transition from q_0 to q_{accept}.

Thus when having accepted a string in L, we start the process at q_0 again thus being able to accept a finite sequence of strings in L. ■

Prefix closure also preserves regularity.

Lemma 1.44 (prefix closure):
Let L be a regular language. Then the prefix closure $L_{\preceq}$ of L and the largest prefix closed sublanguage L' of L are also regular.

Proof: Let $\mathfrak{A}$ be the minimal deterministic finite state automaton for L. A state q is a live state, if there is an accepted string s, such that the accepting run for s passes through q. If we add all live states to F yielding $F_{\preceq}$, the automaton $\mathfrak{A}_{\preceq}$, which is the automaton $\mathfrak{A}$ with $F_{\preceq}$ as set of final states accepts the prefix closure of L. Because if s is an accepted string, then the accepting run passes through a sequence of live states. If $t \preceq s$, then t reaches a live state. Thus $\mathfrak{A}_{\preceq}$ accepts t too.

For an automaton to accept the largest prefix closed sublanguage of L, we remove all states from $\mathfrak{A}$ that are not accept states and also remove all transitions involving such states. The resulting automaton has a prefix closed language. Assume there was a string v in L such that all prefixes belong to L, but that is not accepted by the new automaton. Then the run of the automaton $\mathfrak{A}$ on v must have passed through a state that was not accepting, which is a contradiction to the assumption that every prefix of v also belongs to L.

In particular, if the initial state was not an accept state, ε was not an element of L and thus L did not include any non-empty prefix closed subsets. ■

The last operation we have to be concerned with is convolution of regular languages. Proving that the convolution of two regular languages is regular is a bit tricky, but we only use operations introduced above.

Lemma 1.45 (convolution is regular):
Let n be a natural number and L_i for $1 \leqslant i \leqslant n$ be regular languages over an alphabet A. The convolution $L_1 \otimes \ldots \otimes L_n$ is a regular language over $A^{\otimes n}$.

Proof: Let $\square$ be the blank symbol also used in Definition 1.30 that is not contained in A.

Let $A' := A \cup \{\square\}$ and $L_i' := L_i \cdot \{\square\}^*$. The L_i' are regular languages because they are a concatenation of two regular languages.

Define the projection

$$\pi_i : A^{\otimes n} \to A' : \begin{bmatrix} a^{(1)} \\ \vdots \\ a^{(i-1)} \\ a^{(i)} \\ a^{(i+1)} \\ \vdots \\ a^{(n)} \end{bmatrix} \mapsto a^{(i)}$$

that picks out the n-th component of a symbol in the alphabet $A^{\otimes n}$. We denote also by π_i the extension of π_i to $(A^{\otimes n})^*$.

The languages $\pi_i^{-1}(L_i')$ are regular by Lemma 1.34. We define a new language by intersecting all preimages of the languages L_i'. This yields a regular language by Corollary 1.38:

$$C := \bigcap_{1 \leqslant i \leqslant n} \pi_i^{-1}\left(L_i'\right) \cap A^{\otimes n}.$$

We now show that $C = L_1 \otimes \cdots \otimes L_n$.

For this let $s = s_1 \otimes \cdots \otimes s_n$ be an element of C. By construction, the s_i are elements of L_i' and there is no suffix $\{\square\}^{\otimes n^k}$ for $k > 0$. Thus s is an element of $L_1 \otimes \cdots \otimes L_n$. Conversely let $s_1 \otimes \cdots \otimes s_n$ be an element of $L_1 \otimes \cdots \otimes L_n$. By construction, every $\pi_i^{-1}(s_1 \otimes \cdots \otimes s_n)$ is the set of convolutions of strings with i-th component s_i, and $s_1 \otimes \cdots \otimes s_n$ has no suffix $\{\square\}^k$, thus s is an element of C.

■

Corollary 1.46 (permutation of convolution is regular):
Let L be a regular language. Then $L^{\otimes n}$ is regular and for σ in the group $\mathfrak{S}_n$ of the permutations on n points, the language $\sigma(L^{\otimes n})$ is also regular.

Proof: As L is regular, by Lemma 1.45, the convolution $L^{\otimes n}$ is also regular. Thus, there is a finite state automaton $\mathfrak{A} = (Q, A^{\otimes n}, I, F, \tau)$ that accepts $L^{\otimes n}$. We define an action of $\mathfrak{S}_n$ on τ by

$$\sigma\left(p, \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}, q\right) := \left(p, \begin{bmatrix} a_{\sigma(1)} \\ a_{\sigma(2)} \\ \vdots \\ a_{\sigma(n)} \end{bmatrix}, q\right).$$

Now $\sigma(s) = s_{\sigma(1)} \otimes \cdots \otimes s_{\sigma(n)}$ is an element of $\sigma(L^{\otimes n})$ if and only if $s = s_1 \otimes \cdots \otimes s_n$ is an element of $L^{\otimes n}$ and $\sigma(s)$ is accepted by $\sigma(\mathfrak{A})$; if and only if s is accepted by $\mathfrak{A}$ thus $\sigma(L^{\otimes n})$ is regular. ■

The last construction we need is projection. This construction is also very important to interpret existential quantification in the following paragraph.

Lemma 1.47 (projection is regular):
Let A be an alphabet and let $L_1 \otimes \cdots \otimes L_n$ be a convolution of regular languages over A. Then the projection r_i that removes L_i from the convolution preserves regularity.

Proof: Define the map

$$r_i : A^{\otimes n} \to A^{\otimes(n-1)} : \begin{bmatrix} a_1 \\ \vdots \\ a_{i-1} \\ a_i \\ a_{i+1} \\ \vdots \\ a_n \end{bmatrix} \mapsto \begin{bmatrix} a_1 \\ \vdots \\ a_{i-1} \\ a_{i+1} \\ \vdots \\ a_n \end{bmatrix}.$$

The result follows by Lemma 1.34. ■

Regular languages are useful because they have extensive closure properties. Also because there are many properties of regular languages that can be decided with relatively efficient algorithms. Keep in mind that the often-used procedure to make a given automaton deterministic might blow up the state count exponentially. So even if we can reduce the state count of an automaton to a minimum in $O(n \log n)$, the n to be put into this equation has to be 2^m where m is the state count of the automaton we started with.

Using finite state automata we can decide whether a regular language is empty or whether it is equal to A^*, we can solve the word problem for regular languages and we can decide whether two regular languages are the same. We can enumerate the language of an automaton.

Theorem 1.48 (problems decidable):
Let $\mathfrak{A}$ and $\mathfrak{A}'$ be deterministic finite state automata with languages L and L' respectively. Then the following problems are decidable.

1. *The emptiness problem $L = \emptyset$.*
2. *The universality problem $L = A^*$.*
3. *The finiteness problem $|L| < \infty$ and the co-finiteness problem $|\overline{L}| < \infty$*
4. *The word problem $s \in L$ for $s \in A^*$.*
5. *The equality problem $L = L'$.*

Proof: We can without loss of generality assume $\mathfrak{A}$ as well as $\mathfrak{A}'$ to be minimal. The given problems can be decided as follows.

1. We use depth-first search in $\mathfrak{A}$ to find a path from the initial state of $\mathfrak{A}$ to an accepting state of $\mathfrak{A}$. If there is no such path, then L is empty, if there is, we can give at least one string in L.
2. If $L = A^*$, then the complement $\overline{L}$ is empty, so we can reformulate the universality problem to the emptiness problem for the complement.
3. The language L is finite if and only if the transition graph of the automaton does not contain any cycles. This can be checked easily using graph algorithms.
4. To solve the word problem we just have to check whether the uniquely defined run $\mu(s)$ ends in an accepting state of $\mathfrak{A}$.

5. Because $\mathfrak{A}$ and $\mathfrak{A}'$ are unique and minimal we just have to check for isomorphism of the automata, that is, essentially for isomorphism of the transition graphs.

■

We can enumerate the language of a finite state automaton by using a simple form of depth-first search combined with a backtrack procedure. We search for paths from the start state of an automaton to some accept state and then output the labelling of such a path. This algorithm is particularly interesting in computational group theory for automatic groups, because we have a means to effectively and efficiently enumerate group elements.

To show that a language is not regular, we have to show that there cannot be any finite state automaton that accepts the given language. One way to achieve this is by showing that a language has infinitely many Nerode congruence classes. Another possibility is using the popular pumping lemma. It is included here because it shows a very characteristic property of regular languages and finite state automata.

The pumping lemma only is a sufficient condition, but not necessary. There are non-regular languages that fulfill the pumping lemma.

Theorem 1.49 (pumping lemma):
Let A be an alphabet and L be a regular language over A. Then there exists a constant $n \in \mathbb{N}$, such that for any string $s \in L$ with $|s| > n$, there is a a decomposition of s into three substrings xyz with the three following properties.

- $|y| \geqslant 1$
- $|xz| \leqslant n$
- *xy^iz is in L for all $i \in \mathbb{N}$.*

Proof: By the definition of a regular language there exists a deterministic finite state automaton $\mathfrak{A}$ that decides L. Let s be a string in L with $k = |s| > |Q|$. Let $\mu = q_0s_1q_1 \cdots s_kq_k$ be the accepting run for s on $\mathfrak{A}$. Because $|s| > |Q|$, there is a state p that is visited twice by μ.

Thus μ has the form $q_0s_1q_2 \cdots s_ips_{i+1} \cdots s_jps_{j+1} \cdots s_kq_k$. We can now decompose s into strings $x = s_1 \cdots s_i$, $y = s_{i+1} \cdots s_j$ and $z = s_{j+1} \cdots s_k$. We ensure that $i+k-j < |Q|$, which is the case, if neither x nor y visit a state twice.

We need to show that xy^iz is accepted by $\mathfrak{A}$. We do this by giving an accepting run for this string, $\mu^i = q_0s_1q_2 \cdots s_ip(s_{i+1} \cdots s_jp)^is_{j+1} \cdots s_kq_k$.

■

In the following, we show how finite state automata can be relevant in algebra. This idea was introduced by Khoussainov and Nerode in [KN95] and the theory was subsequently developed by Blumensath and Grädel in [Blu99] and [BG04]. We show that for a subclass of algebraic structures, we can decide for any first order sentence φ whether it holds in the structure by using formal language constructions. We call structures that allow for these methods *automatic structures* or *automatically presentable structures*. We begin by giving a definition for automatically presentable structures.

Definition 1.50 (automatic presentation):
Let $\tau = \{R_1, \cdots, R_n\}$ be a relational signature, and $\mathfrak{S}$ be a τ-structure. Then an automatic *or* regular presentation *for $\mathfrak{A}$ is a tuple*

$$\mathfrak{a} := (A, \pi, L_S, L_\varepsilon, L_{R_1}, \cdots, L_{R_n})$$

satisfying the following conditions:

- *A is an alphabet.*
- *L_S is a regular language over A.*
- *$\pi : L_S \to S$ is a surjective map.*
- *L_ε is a regular subset of $L_S^{\otimes 2}$ with $s_1 \otimes s_2 \in L_\varepsilon$ if and only if $\pi(s_1) = \pi(s_2)$.*
- *L_{R_i} are regular subsets of $L_S^{\otimes n_i}$, where n_i is the arity of R_i and $s_1 \otimes \cdots \otimes s_{n_i} \in L_{R_i}$ if and only if $(\pi(s_1), \cdots, \pi(s_{n_i})) \in R_i^{\mathfrak{S}}$.* □

Given an automatic presentation for a τ-structure and a first order formula in the signature τ, it is possible to determine a set of strings in L_S that encodes elements of the structure, for which the formula holds. In particular we can decide whether a first order sentence holds in $\mathfrak{S}$. We give a very short outline how this is done.

Let $\mathfrak{S} = \left(S, R_1^{\mathfrak{S}}, \cdots, R_n^{\mathfrak{S}}\right)$ be a relational structure and let n_i for $1 \leqslant i \leqslant n$ be the arity of $R_i^{\mathfrak{S}}$. Assume that there exists an automatic presentation $\mathfrak{a} = (A, \pi, L_S, L_\varepsilon, L_{R_1}, \cdots, L_{R_n})$ for $\mathfrak{S}$. Let further φ be an $\mathrm{FO}[\tau]$ formula. The set of variables that occur in φ is finite, thus there exists a number n such that the set of variables occurring in φ is a subset of $\{x_1, \cdots, x_n\}$. We can without loss of generality assume that each variable occurs at most once, and because we use a relational signature, φ does not contain any terms apart from the variables themselves and equalities between variables.

We give a map $\eta^n : \mathrm{FO}[\tau] \to \mathcal{P}\left(L_S^{\otimes n}\right)$ inductively on the structure of the formula, such that for each element $s_1 \otimes \cdots \otimes s_n$ of $\eta^n(\varphi)$, the structure $\mathfrak{S}$ is a model of $\varphi(\pi(s_1), \cdots, \pi(s_n))$.

At this point recall the definition of $\mathrm{FO}[\tau]$ formulae, given in Definition 1.3.

- For formulae $\varphi = R_i\left(x_{j_1}, \ldots, x_{j_{a_i}}\right)$, define a permutation σ_φ by $\sigma_\varphi(k) = j_k$, and $\sigma_\varphi(j_k) = k$ for $1 \leqslant k \leqslant a_i$ and $\sigma_\varphi(k) = k$ for the remaining positions. The convolution $L := L_{R_i} \otimes L_S^{n-a_i}$ is regular and a sublanguage of $L_S^{\otimes n}$ and $\sigma_\varphi(L)$ is the language we want, thus
$$\eta^n\left(R_i\left(x_{j_1}, \ldots, x_{j_{a_i}}\right)\right) := \sigma_\varphi\left(L_{R_i} \otimes L_S^{\otimes n-a_i}\right).$$
- For formulae $\varphi = (x_i = x_j)$, define $\sigma_\varphi(1) := i$ and $\sigma_\varphi(i) = 1$ and $\sigma_\varphi(2) = j$ and $\sigma_\varphi(j) = 2$ and $\sigma_\varphi(k) = k$ for all other points. Then
$$\eta^n(x_i = x_j) := \sigma_\varphi\left(L_\varepsilon \otimes L_S^{\otimes n-2}\right).$$
- For the boolean connectives $\neg$ and $\wedge$ we use the corresponding language constructions complement and intersection. Thus for formulae φ and ψ
$$\eta^n(\neg\varphi) := L_S^{\otimes n} \backslash \eta^n(\varphi)$$
and
$$\eta^n(\varphi \wedge \psi) := \eta^n(\varphi) \cap \eta^n(\psi).$$
- For a formula $\exists x_i \varphi$, we use the projection defined in Lemma 1.47 and a permutation that moves the components into the right position, that is $\sigma(n) = i$ and $\sigma(i) = n$ and $\sigma(j) = j$ for all other indices:
$$\eta^n(\exists x_i \varphi) := \sigma(r_i(\eta^n(\varphi)) \otimes L_S).$$

This leaves the point of a correctness proof open. That is, it has to be shown that $\pi(\eta(\varphi))$ really contains exactly the tuples $(a_1, \cdots, a_n)$ of elements of S, such that $\mathfrak{S} \models \varphi(a_1, \cdots, a_n)$. This is done by induction on the formula. For brevity, we give an example only for the case $R_i(x_j, x_k)$.

$\subseteq$ Let $s_1 \otimes \cdots \otimes s_n$ be an element of $\pi(\eta(R_i(x_j, x_k)))$. The s_i are elements of L_S by definition, and $s_j \otimes s_k$ is an element of L_{R_i}. Thus $(\pi(s_j), \pi(s_k))$ is an element of $R_i^{\mathfrak{S}}$.

$\supseteq$ Let $(a_1, \cdots, a_n)$ be an element of S^n such that (a_j, a_k) is an element of $R_i^{\mathfrak{S}}$. Let $s_1, \cdots, s_n$ be elements of L_S with $\pi(s_i) = a_i$. Then we have $s_j \otimes s_k$ in L_{R_i}, and thus by construction $s_1 \otimes \cdots \otimes s_n$ in $\eta(R_i(x_j, x_k))$.

An immediate consequence of the above construction is that we can decide, given an automatic presentation and an FO$[\tau]$ formula φ, whether φ holds in the presented structure. However, the class of automatic structures is a quite restricted one. More details on automatic presentations of algebraic structures can be found in [Blu99]. The theory of automatic structures is an active area of research, although the research in the theory of automatic structures in general and the research in the theory of automatic groups seems to be seperate.

Corollary 1.51 (automatic structures have a decidable theory):
Let $\mathfrak{S}$ be an automatic structure. Then the FO-theory of $\mathfrak{S}$ is decidable. That is, given a sentence φ in the signature of $\mathfrak{S}$, there is a decision procedure for $\mathfrak{S} \models \varphi$ and $\mathfrak{S} \not\models \varphi$.

Proof: Given an automatic presentation $\mathfrak{a}$ for $\mathfrak{S}$, we can transform φ into a regular language using the map η introduced above. If this language consists of the empty tuple, then $\mathfrak{S} \models \varphi$, otherwise not. ■

1.5 Groups

In this section we look at groups and especially finitely presentable groups. Quite a few of the basic theoretical aspects covered in this section have already been taken care of in Section 1.3, but after all this thesis is about groups so they deserve a section of their own. Groups are commonly referred to as the mathematical abstraction of symmetry. Referring to Section 1.3 we could also say that a group is a set of actions that can not only be sequenced but also be undone. We take a look at the abstract definition of a group.

Definition 1.52 (group):
A group *is an algebraic structure $\mathfrak{G} = (G, \cdot)$ where $\cdot : G \times G \to G$ is a binary operation with the following properties.*

- $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ *for all $a, b, c \in G$.*
- *There exists an element $e \in G$ such that for every $a \in G$, $a \cdot e = e \cdot a = a$ holds.*
- *For each $a \in G$ there is an element denoted $a^{-1} \in G$, such that $a \cdot a^{-1} = a^{-1} \cdot a = e$.*

Furthermore $\mathfrak{G}$ is commutative *or* abelian, *if for all a, b in G the equality $a \cdot b = b \cdot a$ holds.* □

A group is a monoid with inverses for each element. This means, we can refer to results in Section 1.3 and extend these by additional results for groups.

To start off, we look at group homomorphisms, which are defined as one might expect. For groups $\mathfrak{G}$ and $\mathfrak{H}$ a map $\varphi : G \rightarrow H$ is a *group homomorphism* , if $\varphi(g_1 \cdot g_2) = \varphi(g_1) \cdot \varphi(g_2)$ holds for all elements g_1 and g_2 of G. The properties $\varphi(e_{\mathfrak{G}}) = e_{\mathfrak{H}}$ and $\varphi\left(g^{-1}\right) = \varphi(g)^{-1}$ follow easily.

Because we defined the kernel of a monoid homomorphism in Section 1.3, we define the kernel of a group homomorphism as the kernel of the underlying monoid homomorphism. However, we show that this definition is compatible with the usual definition of the kernel of a group homomorphism.

For a group homomorphism $\varphi : \mathfrak{G} \rightarrow \mathfrak{H}$, the kernel of φ is usually defined as the set

$$\ker\varphi := \{g \in G \mid \varphi(g) = e_{\mathfrak{H}}\},$$

that is the set of all elements of $\mathfrak{G}$ that are mapped to the identity of $\mathfrak{H}$.

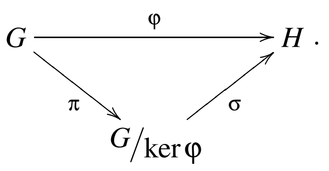
The kernel $\ker\varphi$ of a group homomorphism is a subgroup of $\mathfrak{G}$ with the property that the quotient structure $\mathfrak{G}/\mathfrak{H}$ is a group. These subgroups are called *normal subgroups* and play a central role in the structure theory of groups.

The congruence class $[g]$ of an element g of $\mathfrak{G}$ is the set of elements h of $\mathfrak{G}$ with $\varphi(g) = \varphi(h)$. For groups this condition can be rewritten to $\varphi\left(gh^{-1}\right) = 1$, which is exactly the condition that is used in the usual definition of the kernel of φ.

Definition 1.53 (normal subgroup):
Let $\mathfrak{G}$ be a group. A subgroup $\mathfrak{N}$ of $\mathfrak{G}$ is a normal subgroup, *if there is a group $\mathfrak{H}$ and a group homomorphism $\varphi : \mathfrak{G} \rightarrow \mathfrak{H}$ such that $\mathfrak{N}$ is the kernel of φ.* □

Equipped with this knowledge, we give the homomorphism theorem for groups without any further proof.

Theorem 1.54 (homomorphism theorem for groups):
Let $\mathfrak{G}$ and $\mathfrak{H}$ be groups and $\varphi : \mathfrak{G} \rightarrow \mathfrak{H}$ be a group homomorphism. Then $\varphi = \sigma \circ \pi$, where $\pi(a) = [a]$ is the canonical homomorphism that maps every element of G to the coset $g\ker\varphi$, and σ is an injective group homomorphism with $\sigma(g\ker\varphi) = \varphi(g)$ and $\operatorname{im}\sigma \cong \operatorname{im}\varphi$. In particular, if φ is surjective, we have $\mathfrak{G}/\ker\varphi \cong \mathfrak{H}$. The following diagram commutes:

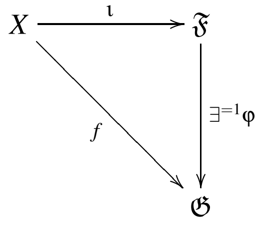


$$\begin{tikzcd} G \arrow[rr, "\varphi"] \arrow[dr, "\pi"'] & & H . \\ & G/\ker\varphi \arrow[ur, "\sigma"'] & \end{tikzcd}$$

□

The free group on a set X is defined analogously to the free monoid on X using a universal property.

Definition 1.55 (free group):
A group $\mathfrak{F}$ is called free *on a subset X of F if it satisfies the universal property, that for any group $\mathfrak{G}$ and any map $f : X \rightarrow G$, there exists a unique group homomorphism $\varphi : F \rightarrow G$ that extends f, that is $f(a) = \varphi(a)$ for all $a \in X$. This can also be expressed using the following commutative diagram, where $\iota : X \rightarrow \mathfrak{F}$ simply denotes the inclusion of X into $\mathfrak{F}$.*



$$\begin{tikzcd} X \arrow[r, "\iota"] \arrow[dr, "f"'] & \mathfrak{F} \arrow[d, "\exists^{=1}\varphi"] \\ & \mathfrak{G} \end{tikzcd}$$

□

Free groups on generating sets of the same cardinality are isomorphic. This result follows directly from Theorem 1.13 because both structures are isomorphic as monoids and both structures are groups.

Theorem 1.56 (free groups):
Let X and Y be sets and let $\mathfrak{F}$ be free on X and $\mathfrak{G}$ be free on Y. Then $\mathfrak{F}$ and $\mathfrak{G}$ are isomorphic if and only if there is a bijective map $f : X \to Y$. □

We denote the free group on a set X by $\mathcal{F}(X)$.

The free group on a set X can also be described as a quotient of a free monoid. We make this point more precise now. The trickiest part is showing that the universal property for a free group holds on the constructed structure, because we have to look at reduced strings under a congruence. By this construction we have as well shown that the free group on a generating set exists and also have a constructive approach to the free group on a set.

Lemma 1.57 (free group as a quotient):
Let X be a set. We take X^{-1} to be a set of the same cardinality as X and a bijective map f from X to X^{-1}. We form the disjoint union $Y := X\dot{\cup}X^{-1}$ and define an involution $\iota : Y \to Y^{-1}$:

$$\iota : Y \to Y : y \mapsto \begin{cases} f(y) & y \in X \\ f^{-1}(y) & y \in X^{-1} \end{cases}.$$

By the use of ι we can formally invert elements of Y. Let $\mathcal{M}(Y)$ be the free monoid on Y. Then

$$\mathcal{F}(X) \cong \mathcal{M}(Y)\big/\approx,$$

where $\approx$ is the congruence on Y generated by $(a\iota(a), \varepsilon)$ and $(\iota(a)a, \varepsilon)$ for all $a \in X$.

Proof: We only need to show the existence of inverses and show that the resulting group is the free group on X, because the structure already is a monoid.

Let $x_1 \cdots x_n$ be an element of $\mathcal{M}(Y)\big/\approx$. The inverse of this element is $\iota(x_n)\cdots\iota(x_1)$. This is because $x_1\cdots x_n\iota(x_n)\cdots\iota(x_1)$ is in the congruence class of ε. Thus it is appropriate to write x^{-1} for $\iota(x)$ for x in Y and also $(x_1x_2\cdots x_n)^{-1} = x_n^{-1}x_{n-1}^{-1}\cdots x_1^{-1}$.

If $\mathfrak{G}$ is a group and f is a mapping $f : X \to G$, we define $f(\iota(x)) := f(x)^{-1}$ and $\varphi(x_1\cdots x_n) := f(x_1)\cdots f(x_n)$. Then it follows easily that φ is a homomorphism of groups and that φ is unique. ■

This seems the right point to choose some conventions concerning group generators. We will always assume group generating sets to be closed under taking inverses; furthermore we will usually take lower case latin letters like x, y, z as generators and will denote formal inverses by the corresponding uppercase letters X, Y, Z. This confronts us with the problem of no longer being able to use X as a generating set, but as we will talk about alphabets anyway we just use the letter A for generating sets after Chapter 1. We also introduced a way to invert elements represented by a string over the generating set. If $v = v_1\cdots v_n$ is a string over the generating set, then its formal inverse is $v_n^{-1}\cdots v_1^{-1}$. That is, the word read backwards and all characters replaced by their formal inverses. Also the formal inverse of a string v represents the inverse of the group element represented by v.

A group $\mathfrak{G}$ is said to be generated by a set X as a semigroup, if and only if there is a surjective semigroup homomorphism $\varphi : \mathcal{S}(X) \to G$, a group $\mathfrak{G}$ is said to be generated by a set X as a group, if and only if there is a surjective group homomorphism $\varphi : \mathcal{F}(X) \to G$. Note that there is a difference between being generated as a group and being generated as a semigroup. For example $(\mathbb{Z},+)$ is generated by 1 as a group but not as a semigroup. A semigroup generating set for $(\mathbb{Z},+)$ is $\{-1,1\}$.

Instead of using equations to generate an equivalence relation on $\mathcal{F}(X)$, we can rewrite any equation $x=y$ that holds in $\mathfrak{G}$ to an equation of the form $xy^{-1}=1$. To form a group presentation we do not need to give equations, but strings over the generating set suffice. Usually, these strings are called relations. This is a clash of names, because relations in the theory of algebraic structures are only loosely connected to relations in group presentations.

We already stated that the kernel of a group homomorphism always is a normal subgroup and describe a normal subgroup of the free group generated by a set of elements of the free group.

Remark 1.58 (describing N):
Let A be a finite alphabet and $R \subseteq \mathcal{F}(A)$. Then the normal subgroup generated by R is equal to the set

$$\overline{R} = \left\{ \prod_{i=1}^{n} w_i r_i^{\pm 1} w_i^{-1} \mid r_i \in R, w_i \in \mathcal{F}(A), n \in \mathbb{N} \right\} \qquad \square$$

We combine the aforementioned statements into the definition of what we want a finitely presentable group to be.

Definition 1.59 (finitely presentable group):
Let $\mathfrak{G}$ be a group and A be a finite set. Then $\mathfrak{G}$ is finitely presentable, *if there is a surjective homomorphism π_A from the free group $\mathcal{F}(A)$ on A to $\mathfrak{G}$, whose kernel is finitely generated as a normal subgroup of $\mathcal{F}(A)$.*

We then present $\mathfrak{G}$ with finite information A and R, where R is a generating set for the kernel of π_A.

$$\mathfrak{G} := \langle A \mid R \rangle \qquad \square$$

When dealing with finitely presented or finitely generated groups, we constantly have to deal with strings v over A such that $\pi_A(v) = g$ for some g in the presented group. We say, that every v in A^* that maps to g via π_A *represents* g. We also use π_A whenever we want to map a string over a generating set to an element of a group and even omit the index when there is no ambiguiety concerning the generating sets.

There are three classical questions associated with finitely presented groups. These questions were first given by the mathematician Max Dehn in [Deh11]. He also gave algorithms to solve these problems for certain classes of groups.

The *word problem* for a finitely presented group $\mathfrak{G} = \langle A \mid R \rangle$ is the question whether for a given word w over A, w is an element of $\overline{R}$. That is, whether w represents the identity in $\mathfrak{G}$.

The *conjugacy problem* is the question whether given two strings v and w over the generators, is there a string s such that $s^{-1}vs =_{\mathfrak{G}} w$.

The *isomorphism problem* is the question whether two finite presentations $\langle A \mid R \rangle$ and $\langle B \mid S \rangle$ present isomorphic groups.

At first glance, at least the word problem looks quite innocent. It is, however, undecidable in general. In [Nov55] P.S. Novikov gives a group presentation that has undecidable word problem. This is a very strong result because it is constructive.

The choice of a set of generators for a group yields a way to visualise the group by a graph, the so called Cayley graph. This visualisation depends on the choice of generators for $\mathfrak{G}$. Geometric group theory as well as combionatorial group theory deal with the properties of Cayley graphs. The notion of the Cayley graph enables us to use geometric and topological arguments for groups. This turns out to be quite useful in the context of automatic groups, as there is a geometric characterisation of a group being automatic.

Definition 1.60 (Cayley graph):
Let $\mathfrak{G}$ be a group generated by a finite subset A of G. Then the Cayley graph

$$\mathfrak{C}(G,A) := \left(V, (E_a)_{a\in A}\right)$$

of $\mathfrak{G}$ with respect to the generating set A is a directed labelled graph with vertex set being the set of all elements of G and edges $(g,h) \in E_a$ between vertices g and h whenever $ga = h$ in $\mathfrak{G}$. □

The Cayley graph of a group is connected and if we assume the generating set to be closed under inverses, we can also think of $\mathfrak{C}(G,A)$ being an undirected graph, because for two vertices g_1 and g_2 with $(g_1,g_2) \in E_x$, there always is an edge $(g_2,g_1) \in E_X$. The Cayley graph gives us a means to talk about distances between group elements. We take the obvious route by defining the distance between two elements of the group as the number of edges in a shortest path that connects the two elements in $\mathfrak{C}(G,A)$. In particular, elements g_1 and g_2 of the group are at distance 1 from each other. if they can be obtained by multiplying one of them with a generator. Certainly this metric depends on the choice of the generating set, but a fundamental result of geometric group theory implies that Cayley graphs of a group with respect to different generating sets are in a certain sense similar.

Definition 1.61 (Cayley graph as metric space):
Let $\mathfrak{G}$ be a group finitely generated by a set A. Let $\mathfrak{C}(G,A)$ be its Cayley graph with respect to the generating set A. As a preparation we define the word length $|g|_A$ *for an element $g \in G$ with respect to the generating set A as follows:*

$$|g|_A := \min\{|v| \mid \pi_A(v) = g, v \in A^*\}.$$

Note that $|v|$ above denotes the length of the string v.
To make $\mathfrak{C}(G,A)$ into a metric space we define a metric $\delta_A : G \times G \to \mathbb{R}_{\geqslant 0}$ as follows:

$$\delta_A : G \times G \to \mathbb{R}_{\geqslant 0} : (g_1,g_2) \mapsto \left|g_1^{-1}g_2\right|_A.$$

Because we are dealing with finitely generated groups, it will be convenient to omit the map π_A when talking about strings over the generating sets. Thus we define for strings u and v in A^*

$$\delta_A(u,v) := \delta_A(\pi_A(u), \pi_A(v)).$$

When it is clear which generating set we are dealing with, we omit the indices. In this context we then denote by $|v|$ the word length of v unless stated otherwise. □

In the theory of automatic groups we will also have to deal with another metric involving a language that maps onto $\mathfrak{G}$. Many authors do not seem to stress this enough, thus we watch closely.

Definition 1.62 (uniform distance):
Let $\mathfrak{G}$ be a group finitely generated by the set A and let $L \subset A^*$ be a language that maps surjectively onto G via the restriction of π to L. Let v and w be strings in L. The uniform distance $\delta_{A,\preceq}(v,w)$ *of v and w is defined as*

$$\delta_{A,\preceq}(v,w) := \max\{\delta_A(\pi(v[t]), \pi(w[t])) \mid t \in \mathbb{N}\},$$

where δ_A is the metric defined in 1.61. The uniform distance is not a metric but a pseudometric, because two distinct strings might have distance zero. □

As conclusion we define a class of finitely presented and geometrically motivated groups that have very nice properties. For example all groups in this class are automatic. The triangle groups we will be dealing with later are a subclass of this class of groups. For this we need the definition of a hyperbolic space. This definition involves a few notions from geometry which we only give intuitively and for Cayley graphs.

A *geodesic* is a shortest path that realises a distance, that is for two elements u and v of a metric space at distance c, there is a path of length c that connects u and v. Cayley graphs are always geodesic, because for two elements g_1 and g_2 of $\mathfrak{G}$ the path labelled by the representative of $g_1^{-1}g_2$ is a geodesic and it precisely realises the distance between g_1 and g_2 as given in Definition 1.61. A *triangle* consists of three vertices in the Cayley graph, connected by geodesics. A triangle is said to be α-thin, if there is a constant $\alpha \in \mathbb{R}_{\geqslant 0}$ such that the distance between any point on one of the triangle's sides to the union of the two other sides is bounded by α.

Definition 1.63 (word hyperbolic group):
Let $\mathfrak{G}$ be a group finitely generated by the set A. Then $\mathfrak{G}$ is said to be word hyperbolic *with respect to the generating set A if there is a constant α, such that every triangle in the Cayley graph $\mathfrak{C}(G,A)$ is α-thin.* □

Word hyperbolicity is a property of many groups studied in geometric group theory. It can be shown, that word-hyperbolicity does not depend on the choice of generators for $\mathfrak{G}$. This theorem can be looked up in [dlH00]. Also, word hyperbolicity is very rich in the sense that there are many characterisations of word hyperbolic groups. Among them is a purely language theoretic characterisation of word hyperbolicity by context freeness of the multiplication table. This remarkable result is due to Robert Gilman and can be found in [Gil].

We should now be equipped with enough theory to continue with the theory of automatic groups.

2 Automatic Groups

”Let him that hath understanding
find the order of the group $G =$
$\langle\, x,y,z,t \mid x^3y^7, y^4x^7, z^3t^5, t^4z^6, [x,z] \,\rangle$”

(D.L. Johnson)

In this chapter we introduce automatic groups. Automatic groups are finitely presentable groups, whose Cayley graph allows for an automatic presentation. As a consequence these groups have nice computational properties, such as an efficiently solvable word problem and solvable conjugacy problem. We keep in mind that these problems are not decidable in general. From a researcher’s perspective, automatic groups are interesting, because there are quite a few open questions that need to be answered. To the knowledge of the author, it is currently unknown whether the isomorphism problem for automatic groups is decidable or if every automatic group admits a biautomatic structure, to name two examples. Automatic groups are also important, because they provided a tool to prove a case of Thurston’s conjecture, which in turn implies the Poincaré conjecture.

Automatic groups were the first algebraic structures that were found to allow for an automatic presentation and were introduced by David Epstein and John W. Cannon in their work preceding the book [EPC$^+$92]. This book collects the work of the respective authors to give a complete reference to the topic. A somewhat more accessible account to the computational methods for automatic groups can be found in [HEO05].

It is remarkable how geometric group theory and the theory of formal languages connect in this topic.

2.1 Automatic Presentations for Groups

The idea of an automatic presentation was introduced in Section 1.4. We now specialise this idea a bit and give the definition of an automatic presentation for a finitely presented group. We do not represent the graph of multiplication in the group as a whole but rather decompose it into relations that represent right or left multiplication by a generator. Thus we give an automatic presentation for the Cayley graph of $\mathfrak{G}$ with respect to a generating set A.

Definition 2.1 (automatic presentation):
Let $\mathfrak{G}$ be a group, finitely generated by A as a monoid. Then

$$\mathfrak{a}(\mathfrak{G}) = \left(A, \pi, \mathfrak{W}, \mathfrak{M}_\varepsilon, (\mathfrak{M}_x)_{x\in A}\right)$$

consisting of the alphabet A, a finite state automaton $\mathfrak{W}$ over A, a finite state automaton $\mathfrak{M}_\varepsilon$ over $A \otimes A$, finite state automata $\mathfrak{M}_x$ over $A \otimes A$ for all x in A and a map $\pi : L(\mathfrak{W}) \to G$ is an automatic presentation *for $\mathfrak{G}$, if the restriction of $\pi : A^* \to G$ to $L(\mathfrak{W})$ is surjective and if for x in $A \cup \{\varepsilon\}$, the string $v \otimes w$ is accepted by $\mathfrak{M}_x$ if and only if vx as well as w are accepted by $\mathfrak{W}$ and $\pi(vx) = \pi(w)$.*
We call a group that allows for such a presentation automatically presentable *or* automatic. □

There is a very important point to be stressed here. In this thesis we take π to be the restriction of the monoid homomorphism $\pi : A^* \to G$ to $L(\mathfrak{W})$. The general definition for arbitrary automatic structures does not include this restriction. Also we present right multiplication by a generator as regular relation. One can also look at automatic presentations for groups that present the graph of the multiplication as a whole. This point is often a source of errors and misunderstandings. This thesis will exclusively deal with the given definition of an automatic group, which might be called the classical or the canonical one.

There are groups that have an automatic presentation the general sense but not in the classical sense. For example, the Heisenberg group given in Section 2.4 is not automatic in the sense of Section 2.1. A proof of this can be found in Chapter 7 of [EPC$^+$92] where it is proven that infinite nilpotent groups are not automatic. But there is an automatic presentation in the sense of Definition 1.50. This is proven in [BG04] and the proof is surprisingly simple.

If $\mathfrak{a} = \left(A, \pi, \mathfrak{W}, \mathfrak{M}_\varepsilon, (\mathfrak{M}_x)_{x\in A}\right)$ is an automatic presentation, we call $\mathfrak{W}$ the *word acceptor*, $\mathfrak{M}_\varepsilon$ the *equality recogniser* and the $\mathfrak{M}_x$ *multiplier automata*. If $\mathfrak{W}$ accepts exactly one string for each element of $\mathfrak{G}$, then $\mathfrak{W}$ is called a *unique word acceptor* and the presentation is called *injective* because π is then injective.

If for v and w in $L(\mathfrak{W})$, the equality recogniser $\mathfrak{M}_\varepsilon$ accepts $v \otimes w$, then v equals w as elements of $\mathfrak{G}$. Unless $\mathfrak{W}$ is unique, v and w need not be equal as strings over A. There may be infinitely many representatives for one element of $\mathfrak{G}$ in $L(\mathfrak{W})$. Because it is a bit awkward to write $\pi(v) = \pi(w)$, we write $v =_{\mathfrak{G}} w$, which means that v is equal to w when interpreted as elements of $\mathfrak{G}$. Also, when talking about automatic groups and no ambuiguity arises, we take $\mathfrak{a}$ to be the automatic presentation, $\mathfrak{W}$ to be the word acceptor, $\mathfrak{M}_\varepsilon$ and $\mathfrak{M}_x$ to be the multiplication automata for $x \in A$ and W, M_ε and M_x denote the languages of the respective automata.

We could equivalently define a notion of a regularly presentable group and put in regular languages in place of the finite state automata. As we will be dealing with algorithms for automatic groups, automata are a better choice.

We will now take a look at the properties of automatically presentable groups and the boundaries of the concept of automatic groups. As we have established in Section 1.4, regular languages allow for effective algorithms and decision procedures. Given an automatic presentation, we can decide whether a string s over A represents an element of the group checking whether the word acceptor accepts it. We can enumerate the language $L(\mathfrak{W})$ using depth first or breadth first search on the word acceptor, effectively enumerating group elements. We will also give a method to find a shortest representative for a group element g in $L(\mathfrak{W})$ if we are given an arbitrary representative of g. Using the equality automaton $\mathfrak{M}_\varepsilon$ we can decide whether two strings represent the same element of $\mathfrak{G}$, thus effectively solving the word problem. The conjugacy problem needs a bit more work. Using the multiplier automata $\mathfrak{M}_x$, we can multiply elements of the group by a generator, thus we are able to multiply group elements.

For an automatic group $\mathfrak{G}$, an automatic presentation $\mathfrak{a}(\mathfrak{G})$ really is an automatic presentation of the Cayley graph of $\mathfrak{G}$ with respect to the generating set A. The automaton $\mathfrak{W}$ accepts labels of paths in the Cayley graph, and the $\mathfrak{M}_x$ accept the labelled edge relations in the Cayley graph. In Definition 1.61 we introduced a natural notion of distance in the Cayley graph, and we can now show that the Cayley graph of an automatic group has special geometric properties.

If two strings v and w over A represent two group elements that differ only by the multiplication by a generator, that is if $vx =_{\mathfrak{G}} w$, the two paths starting at the identity element of $\mathfrak{G}$ labelled with v and w respectively will always be close together, they are said to *fellow travel*. This is because of the restrictions that apply to regular languages, resulting from finite state without any kind of memory.

We have to consider two strings v and w, such that $v \otimes w$ is accepted by $\mathfrak{M}_x$ for some $x \in A \cup \{\varepsilon\}$. We have to consider all prefixes $v[t]$ and $w[t]$ for $t \in \mathbb{N}$, and look at the distance between the elements represented by $v[t]$ and $w[t]$ in the Cayley graph. Note, that we can always take prefixes of strings and formally invert strings in a language, but this might lead us outside the language.

Definition 2.2 (fellow traveller property):
Let $\mathfrak{G}$ be a group generated by A as a monoid and let $W \subseteq A^*$ be a language of representatives for the elements of $\mathfrak{G}$. The language W is said to have the fellow traveller property, *if there is a constant $k \in \mathbb{N}$, such that for all elements v and w of W with $vx =_{\mathfrak{G}} w$ for $x \in A \cup \{\varepsilon\}$, the uniform distance $\delta_{A,\preceq}(v,w)$ between v and w is bounded by k. The constant k is usually called the* Lipschitz constant. □

A fundamental property of automatic groups is that the language of the word acceptor for any automatic presentation has the fellow traveller property. This will allow us to express multiplication by a generator as a regular relation.

Theorem 2.3 (automatic implies fellow traveller property):
Let $\mathfrak{G}$ be an automatic group with automatic presentation $\mathfrak{a}$. Then $L(\mathfrak{W})$ has the fellow traveller property and the constant $k \in \mathbb{N}$ can be chosen to be $2m+1$ where m is the maximum over the count of states of all multiplication automata in the automatic presentation.

Proof: Let m be the maximum over the number of states of all automata $\mathfrak{M}_x$ for $x \in A \cup \{\varepsilon\}$.

We have to show that for all strings v and w in W with $vy =_{\mathfrak{G}} w$ the distance $\delta(A, \preceq)(v,w)$ is bounded by $2m+1$, that is we have to show that $\delta(v[t], w[t])$ is bounded by $2m+1$.

Because $vy =_{\mathfrak{G}} w$, the automaton $\mathfrak{M}_y$ accepts $v \otimes w$. Thus there is an accepting run $\mu(v \otimes w)$ of $\mathfrak{M}_y$ on $v \otimes w$. Because every state in the run is live, for $v[t] \otimes w[t]$ there exists a string $v' \otimes w'$ of length shorter than or equal to m such that $v[t]v' \otimes w[t]w'$ is accepted by $\mathfrak{M}_y$, thus $v[t]v'y =_{\mathfrak{G}} w[t]w'$.

Therefore $v[t]^{-1}w[t] =_{\mathfrak{G}} v'yw'^{-1}$, thus the distance between $v[t]$ and $w[t]$ in the Cayley graph is bounded by the length of $v'yw'$, which we know to be less than $2m+1$.

■

We now take a closer look at what it means for a group to have a language of representatives that has the fellow traveller property and work towards a characterisation of automatic groups. Following the idea of looking at prefixes of strings and distances in the Cayley graph we define word differences as follows.

Definition 2.4 (word difference):
Let $\mathfrak{G} := \langle A \mid R \rangle$ be a finitely presented group and let L be a language over A that maps surjectively onto G. Let v and w be strings in L. Then the set $D(v,w)$ of word differences associated with v and w is defined as

$$D(v,w) := \left\{ \pi\left(v[t]^{-1}w[t]\right) \mid t \in \mathbb{N} \right\},$$

and is a subset of G.

For a set P of pairs of strings, we define the set $D(P)$ of word differences associated with P as the union of the word differences associated with the pairs in P. We define D_x for each generator x in A as follows

$$D_x := \{ D(v,w) \mid vx =_{\mathfrak{G}} w \}.$$

We further define the set D to be the union of all D_x for x in A.

$$D := \bigcup_{x \in A \cup \{\varepsilon\}} D_x$$

□

The sets D_x and D defined in Definition 2.4 are finite in the case that the language has the fellow traveller property because of Theorem 2.3. In particular, if $\mathfrak{G}$ is automatic, the set D is finite. Word differences play a vital role in the construction of automatic presentations for finitely presented groups, because we will use the set D as state set for a finite state automaton.

In the following we look at methods to derive an automatic presentation for a group $\mathfrak{G}$ from the knowledge that the group can be regularly generated and has the fellow traveller property. In conclusion we prove that the property of a group being automatic can be characterised as the group being regularly generated and the word language having the fellow traveller property. We begin by defining a finite state automaton based on the set of word differences defined in Definition 2.4.

Definition 2.5 (word difference automaton):
Let $\mathfrak{G} = \langle A \mid R \rangle$ be a finitely presented group and

$$\mathfrak{Z} = \left(Q, A^{\otimes 2}, \{q_0\}, Q, \tau\right)$$

a deterministic finite state automaton over $A^{\otimes 2}$. We call $\mathfrak{Z}$ a word difference automaton *for $\mathfrak{G}$ if there is a map $f : Q \to G$ with $f(q_0) = 1$ and $f\left(\tau\left(q, \begin{bmatrix} a \\ b \end{bmatrix}\right)\right) =_{\mathfrak{G}} a^{-1} f(q)\, b$ for all $q \in Q$ and $\begin{bmatrix} a \\ b \end{bmatrix} \in A^{\otimes 2}$.*□

In general we cannot tell whether a word difference automaton exists for a given finitely presented group $\mathfrak{G}$. But if we know how the Cayley graph looks like, we are sometimes able to compute a word difference automaton. If we are, we can also give multiplication automata for $\mathfrak{G}$. We first show how this is accomplished and afterwards make explicit how a word difference automaton can actually be computed.

If $\mathfrak{Z}$ is a word difference automaton for a group $\mathfrak{G}$, the states of $\mathfrak{Z}$ represent word differences of strings over the generators. This can be seen by the following example. Let $v \otimes w$ be a string over $A^{\otimes 2}$. There is a unique run $\mu(v \otimes w)$ of $\mathfrak{Z}$. Because $f(q_0) = 1$ and $f\left(\tau\left(q, \begin{bmatrix} a \\ b \end{bmatrix}\right)\right) =_{\mathfrak{G}} a^{-1} f(q)\, b$, we effectively compute the word difference of v and w. Thus, if $f(q) =_{\mathfrak{G}} x$ for $q \in Q$ and $x \in A$, then $vx =_{\mathfrak{G}} w$. This also enables us to give multiplication automata constructively, if we are able to compute a word difference automaton for $\mathfrak{G}$.

To give a constructive version of a word difference automaton, we consider the set D of word differences defined in Definition 2.4 assuming it to be finite. We take D to be the state set for our word difference automaton. The set of transitions consists of tuples $\left(p, \begin{bmatrix} a \\ b \end{bmatrix}, q\right)$ for $a^{-1}pb =_{\mathfrak{G}} q$. We take ε to be the start state, which certainly exists as a word difference because we assume the generating set to be closed under inverses. The set of accept states consists of all states in D. This is not entirely right, because this word difference automaton might accept strings $v \otimes w$ such that v or w have padding symbols as proper infix. This can be taken care of by construction of an automaton, that accepts the language of $\mathfrak{Z}$ intersected with $A^* \otimes A^*$. Such an automaton exists because this intersection is regular.

Theorem 2.6 (word difference automaton and multipliers):
Let $\mathfrak{G} := \langle A \mid R \rangle$ be a finitely presented group. If there is a regular language W over A, such that $\pi : W \to G$ is surjective and W has the fellow traveller property, we can construct multiplication automata $\mathfrak{M}_x$ for $x \in A$ thus completing the automatic presentation for $\mathfrak{G}$.

Proof: Because W has the fellow traveller property, the set D defined in Definition 2.4 is finite and thus there is a word difference automaton for $\mathfrak{G}$ as described above.

To construct a multiplication automaton $\mathfrak{M}_x$ from $\mathfrak{Z}$, we first change the set F of accept states to contain only states q with $f(q) =_{\mathfrak{G}} x$. We call the resulting automaton $\mathfrak{Z}_x$ and denote its language by Z_x as usual. This automaton might still accept strings $v \otimes w$ over $A \otimes A$ such that v or w are not elements of W. This problem can be adressed by forming the intersection $Z_x \cap W \otimes W$. This language still is regular because of Lemma 1.45 and Corollary 1.38, thus we can construct an automaton for it and call it $\mathfrak{M}_x$.

It is now necessary to prove that $\mathfrak{M}_x$ is correct. That is, $\mathfrak{M}_x$ accepts $v \otimes w$ if and only if v and w are accepted by $\mathfrak{W}$ and $vx =_{\mathfrak{G}} v$.

For this let $v \otimes w$ be accepted by $\mathfrak{M}_x$. Then $v \otimes w$ is by construction an element of $W \otimes W$ as well as being accepted by $\mathfrak{Z}_x$, thus v and w are accepted by $\mathfrak{W}$ and $vx =_{\mathfrak{G}} w$.

Conversely, let v and w be elements of W with $vx =_{\mathfrak{G}} w$. Then $\mathfrak{Z}_x$ accepts $v \otimes w$ by construction and thus $\mathfrak{M}_x$ does too. ■

We are now able to characterise automatic groups by the fellow traveller property.

Theorem 2.7 (regularly generated and fellow travellers):
Let $\mathfrak{G}$ be a group with generating set A and W be a regular language over A that maps surjectively onto G. Then W has the fellow traveller property if and only if $\mathfrak{G}$ is automatic with word acceptor $\mathfrak{W}$ having language W.

Proof: This statement follows directly from Theorem 2.3 and Theorem 2.6. ■

This result has, however, to be taken with some caution. To constructively know the word difference system, we have to know the Cayley graph of $\mathfrak{G}$ at least for a finite neighbourhood of the identity. That is, we need to solve the word problem.

An algorithm that actually attempts to compute automatic presentations for finitely presented groups will be presented in Chapter 3. This algorithm relies heavily on the use of a system of axioms for the class of automatic presentations for a finitely presented group which will be given in Section 2.3.

2.2 Properties of Automatic Presentations

In this section we take a short look at further properties of automatic presentations and automatic groups.

The theory introduced in Section 2.1 depends on a finite generating set for the group to be examined. One might expect that the very property of being automatic depends on the choice of the generating set. Luckily this is not the case. The following theorem justifies the notion of the group being automatic and not only a presentation of it being automatic.

We cannot make any statement statement about the quality of the automatic structure: A group might have an easily handled **ShortLex** automatic structure with respect to some set of generators and very complex ones with respect to other sets of generators. There are, until now, no results that allow for a better understanding of the situation.

In the following let $\mathfrak{G}$ be a group and A and B be generating sets for $\mathfrak{G}$. Thus there are surjective maps $\pi_A : A^* \to G$ and $\pi_B : B^* \to G$. The first lemma is just an extension of the fellow traveller property.

Lemma 2.8 (moving strings):
Let $\mathfrak{G}$ be automatic with Lipschitz constant k. Let u, v and w be elements of W and let v be of length $c > 0$. Let further $uv =_{\mathfrak{G}} w$. Then $\delta_{A,\preceq}(u,w)$ is less than kc.

Proof: Assume $v = v'x$ with $x \in A$. Then $uv'x =_{\mathfrak{G}} w$ and $\delta_{A,\preceq}(uv',w)$ is bounded by k by the fellow traveller property. By induction $\delta_{\preceq}(u,w)$ is less than kc. ■

The following lemma shows that we can add or remove elements representing the identity from generating sets without changing the property of the group being automatic. This lemma and the following corollary provide a technical tool used in the proof of *Theorem* 2.11

Lemma 2.9 (change of generators I):
Let $\mathfrak{G}$ be automatic and $\mathfrak{a} = \left(A, \pi, \mathfrak{W}, \mathfrak{M}_\varepsilon, (\mathfrak{M}_a)_{a\in A}\right)$ be an automatic presentation for $\mathfrak{G}$. If $B = A \cup \{e\}$ or $A = B \cup \{e'\}$ where e and e' are representatives of the identity element of $\mathfrak{G}$, then there is an automatic presentation $\mathfrak{b}$ for $\mathfrak{G}$ with generating set B.

Proof: We first tackle the easy direction. If $B = A \cup \{e\}$, then we just make $\mathfrak{W}$ into an automaton over B that does not accept strings that contain e. This can easily be achieved by adding transitions labelled with e from each state to a failure state.

It seems tempting to try the same thing when $A = B \cup \{e'\}$. This does not work, because deleting characters from strings might change the word metric. But we can nonetheless prove that there is a regular language over B that maps surjectively onto G and has the fellow traveller property. We note that we can without loss of generality assume B to be not empty, because if it were, we would be dealing with the trivial group. We have to think of a way to delete e's from strings in which it occurs without changing the word metric.

For this, let v be a string of length n that represents the identity. If we replace every m-th occurrence of e' with v and delete all other occurrences of e', the result is a regular language and the fellow traveller property is not affected.

We show that the constructed language is regular by giving an automaton $\mathfrak{W}_B$ that accepts it.

Let $\mathfrak{W} = (Q, A, (q_0), F, \tau)$ and $\mathfrak{W}_B = (Q', B, (q_0'), F', \tau')$. Define $Q' := Q \times \{0, \ldots, m-1\}$ and $q_0' := (q_0, 0)$ and $F' := \{(q,i) \mid q \in F, i \in \{0, \ldots, m-1\}\}$. For each transition (p,x,q) in τ with $x \neq e'$, we add the transitions $((p,i), x, (q,i))$ for $0 \leqslant i < m$ to τ' and for transitions (p,e',q) we add a transition $((p,0), z, (q, m-1))$ and transitions $((p,i), \varepsilon, (q, i-1))$ for $1 \leqslant i < m$. We note that is does not matter that z is a string. We can as well choose a symbol z' that is not in A to replace e' and later map that symbol to the string z.

We show that $\mathfrak{W}_B$ accepts exactly the language described above. For this, let s be a string in $L(\mathfrak{W})$. If s does not contain e', it is surely accepted by $\mathfrak{W}_B$, because we can make the run of $\mathfrak{W}$ on s into a run on $\mathfrak{W}_B$ by making each transition (p,x,q) into a transition $((p,0),x,(q,0))$. If s contains a number of e's the construction replaces every m-th occurrence with z, let the result of this be s'.

Suppose the run of $\mathfrak{W}$ on s of length n was

$$q_0 s_1 q_1 s_2 \dots s_{i-1} q_i e' q_{i+1} s_{i+1} \dots s_j q_j e' q_{j+1} \dots s_n q_n,$$

where i and j are natural numbers such that $s_1 \dots s_{i-1}$ and $s_{i+1} \dots s_{j-1}$ do not contain any e's. The correspoding run of $\mathfrak{W}_B$ on s' then looks as follows:

$$(q_0,0)\, s_1\, (q_2,0)_2 \dots s_{i-1}\, (q_i,0)\, z\, (q_{i+1}, m-1)\, s_{i+1} \dots s_{j-1}\, (q_j, m-1)\, \varepsilon\, (q_{j+1}, m-2) \dots s_n\, (q_n, i).$$

In the theory of nondeterministic computation what happens when using the ε-transition this is often referred to as *guessing* a position where e' occurred in the original string. Effectively the automaton just tries every possible combinations of substrings in $L(\mathfrak{W})$ containing no symbol e'. If $\mathfrak{W}_B$ accepts a string, it might be longer than the original string, but the increase in length is bounded by $m-1$.

Because the difference in length is bounded, the conditions of Theorem 2.7 are satisfied. Thus we have proven the desired result. ■

As a final tool we need to map generators from one generating set to strings of constant length over the other generating set. This is now fairly easy to accomplish, because we can always add a representative of the identity element to the generating set and then fill up strings.

Corollary 2.10 (uniform map):
There is a constant $c \in \mathbb{N}$ such that there is a map $f : A \to B^*$ with $\pi_A(x) = \pi_B(f(x))$ and $|f(x)| = c$ for all x in A.

Proof: Because A as well as B are generating sets for $\mathfrak{G}$, we can for each $\pi_A(x)$ choose an element u_x of $\pi_B^{-1}(\{\pi_A(x)\})$. The set A is finite, thus we choose the constant c to be the maximum of lengths of the u_x for all x in A. Because of Lemma 2.9, we can assume A as well as B to contain a representative of the identity, thus we can fill up the strings u_x for each x in A to a string of length c and call it u'_x. Then we define

$$f : A \to B^* : x \mapsto u'_x.$$

In conclusion we prove that the property of being automatic is invariant under change of generators.

Theorem 2.11 (change of generators II):
Let $\mathfrak{G}$ be a group and A and B be monoid generating sets for $\mathfrak{G}$. Then $\mathfrak{G}$ is automatic with respect to A if and only if $\mathfrak{G}$ is automatic with respect to B.

Proof: We show the result by showing that we can change generators without destroying regularity and the fellow traveller property. Let $\mathfrak{G}$ be automatic with respect to A, let W_A be the language of the word acceptor and let k be the Lipschitz constant for α.

We want to find a regular language W_B over B that has the fellow traveller property and maps surjectively onto G.

By Corollary 2.10, we have maps $f : A \to B^*$ and $g : B \to A^*$ with $|f(x)| = c$ for all x in A and a constant c and $|g(y)| = c'$ for all y in B and a constant c' and by Lemma 1.34, the extension of f to a monoid homomorphism maps W_A to a regular language. We define $W_B := f(W_A)$. Because A is a generating set for $\mathfrak{G}$, the map $\pi_B \circ f$ is surjective.

We have to show that W_B has the fellow traveller property. That is, for two strings u and v in W_B with $ub =_{\mathfrak{G}} v$, the uniform distance $\delta_{B,\preceq}(u,v)$ is bounded by a constant k'. For this let u and v be strings in W_B and y an element of B such that $uy =_{\mathfrak{G}} v$ and let n' be some natural number. We have to show, that $\delta_B(u[n'],v[n'])$ is bounded by a constant.

We let $s = g(u)$ and $t = g(v)$ in W_A. Note, that $\pi_A(s) =_{\mathfrak{G}} \pi_B(u)$ and $\pi_A(s) =_{\mathfrak{G}} \pi_B(u)$. Because $\delta_B(u,v) \leqslant 1$, we know that $\delta_A(s,t) \leqslant c'$ and by Lemma 2.8, $\delta_{\preceq,A}(s,t) \leqslant kc'$.

By construction there is a number n such that $\delta_B(f(s[n]),u[n']) \leqslant \frac{c}{2}$ and $\delta_B(f(t[n]),v[n']) \leqslant \frac{c}{2}$. Because W_A has the fellow traveller property, we know that $\delta_A(s[n],t[n])$ is bounded by kc'. We also know, that $\delta_B f(s[n]), f(t[n])$ is less than kcc', thus in conclusion we can bound $\delta_{B,\preceq}(u,v)$ by $c + kcc'$.

■

A really interesting question is which regular subsets of A^* we can use as language of a word acceptor. This is also one of the topics in the theory of automatic groups that has some interesting open questions. One example is the question, in which cases the language of all shortest representatives for elements of $\mathfrak{G}$ is regular. In general it is not regular, but there are very important classes of groups where the language consisting of shortest representatives is regular.

If a group is automatic with a word acceptor that accepts the language consisting of all geodesic strings, the group is called strongly geodesically automatic, and if there is some language containing a geodesic for each element of $\mathfrak{G}$, the group is called weakly geodesically automatic.

Another important case is the case of a **ShortLex** automatic structure. If a group allows for a word acceptor that only accepts **ShortLex** minimal geodesics the group is called **ShortLex** automatic.

We take a short look at what we can do with languages that are the language of a word acceptor for a group. We can restrict to regular subsets of a given language as long as π stays surjective. We keep in mind that regular languages are not closed unter taking arbitrary subsets.

Lemma 2.12 (restriction):
Let $\mathfrak{G}$ be an automatic group and let W the language of a word acceptor for $\mathfrak{G}$. Then every regular sublanguage L' of W that maps surjectively onto G, also gives an automatic presentation for $\mathfrak{G}$.

Proof: As $\mathfrak{G}$ is automatic, the fellow-traveller property holds for W, and thus also for L'. By Theorem 2.7 we can construct multiplier automata $\mathfrak{M}_x$ for x in $A \cup \{\varepsilon\}$ and have an automatic structure.
■

We can make any regular language W into a regular prefix-closed language. If W maps surjectively onto a finitely presented group $\mathfrak{G}$, then $W_{\preceq}$ also does and we construct an automatic presentation with a prefix closed language.

Lemma 2.13 (prefix closure):
Let $\mathfrak{G}$ be an automatic group and W be the language of the word acceptor of an automatic presentation of $\mathfrak{G}$ and k be the Lipschitz constant for this presentation. Then there is an automatic presentation for $\mathfrak{G}$ that has the prefix closure $W_{\preceq}$ of W as language of its word acceptor.

Proof: The prefix closure $W_{\preceq}$ of W is regular by Lemma 1.44. We need to verify the fellow traveller property for $W_{\preceq}$. For this let v and w be elements of $W_{\preceq}$ with $vx =_{\mathfrak{G}} w$. Because v as well as w are elements of $W_{\preceq}$, there are strings s and t of bounded length such that vs and wt are elements of W. The lengths of s and t are bounded by the state count c of $\mathfrak{W}$.

It follows that $\delta_A(vs, wt)$ is at most $2c+1$ and thus that $\delta_{A,\preceq}(vs, wt) \leqslant k(2c+1)$. Therefore also $\delta_{A,\preceq}(v, w) \leqslant k(2c+1)+2c$.

■

We can turn any automatic structure into one that also has unique representatives for elements of $\mathfrak{G}$. Do not let this result fool you into the perception that every group has a **ShortLex**-automatic presentation, it only states that we can choose **ShortLex**-minimal representatives from the language of the word-acceptor of a previously determined automatic presentation. A **ShortLex** automatic group needs to have a **ShortLex**-automatic presentation where the minimal representatives are the ones taken from A^*, not some sublanguage of A^*.

Lemma 2.14 (uniqueness):
Let $\mathfrak{G}$ an automatic group and let W be the language of a word acceptor for $\mathfrak{G}$. Then there is an automatic presentation of $\mathfrak{G}$ that has a word acceptor that accepts a unique string from W for each group element and the equality recogniser only accepts strings $v \otimes v$ for v in W.

Proof: Let $\sqsubseteq$ be any total order on W that can be automatically presented. Every subset of W thus has a least element under this order. We form the subset W' of W that is regular, because we chose $\sqsubseteq$ to be automatic. Let

$$W' := \{s \in W \mid \forall t\,(s =_{\mathfrak{G}} t \rightarrow s \sqsubseteq t)\}.$$

The language W' is a sublanguage of W and maps surjectively onto G. For existence we need at least one total order that is automatic. The **ShortLex** order is a total and automatic order on strings over an alphabet. ■

Another nice property of automatic groups is that representatives of one group element can only be of bounded length difference. This is an application of the pumping argument.

Lemma 2.15 (bounded length difference):
Let $\mathfrak{G}$ be an automatic group. There is a constant $N \in \mathbb{N}$ such that for any v in W and g in $\mathfrak{G}$ with $\pi(v)a = g$ for some a in $A \cup \{e\}$, we have the following situatuion

- *there is another string w in W with $\pi(w) = g$ of length less than or equal to $|v| + N$ and*
- *if there is a w' in W with $\pi(w) = g$ of length greater than $|v| + N$, there are infinitely man representatives for g in W.*

Proof: Take N to be larger as the maximum state count of all automata involved in $\mathfrak{a}$. The rest follows by a simple pumping argument. ■

Up until now we only dealt with right multiplication by a generator. For a group it should not matter whether we look at left or right multiplication. Thus we briefly look at what happens if we also want to represent left multiplication by a generator as regular relation. A biautomatic presentation of a group enables us to solve the conjugacy problem. It is currently unknown whether every automatic group is biautomatic.

Definition 2.16 (biautomatic):
Let $\mathfrak{G}$ be an automatic group and let W be the language of a word acceptor. If the language W^{-1} of formal inverses of strings in W is the language of the word acceptor of an automatic presentation for $\mathfrak{G}$, then $\mathfrak{G}$ is called biautomatic. □

Biautomaticity has a characterisation that is similar to Theorem 2.7 involving the left and right multiplication by generators.

As conclusion to this section we show that word-hyperbolic groups are automatic.

Theorem 2.17 (word hyperbolic groups are automatic):
Let $\mathfrak{G}$ be finitely generated by a set A and word-hyperbolic. Then $\mathfrak{G}$ is automatic with a prefix closed language consisting of geodesics.

Proof: Let u and v be geodesic strings over A such that $\delta_A(u,v) \leqslant 1$. Then u and v are two sides of a geodesic triangle in the Cayley graph of $\mathfrak{G}$ with respect to A and thus because every triangle is thin, $\delta_{A,\preceq}$ is bounded.

This leaves the question that the language of geodesics is in fact regular. We refer to [EPC$^+$92], Theorem 3.2.2 for the proof. ■

2.3 Axioms

Given a finitely presented group $\mathfrak{G} = \mathrm{Mon}\langle A \mid R \rangle$ we want to compute an automatic presentation for $\mathfrak{G}$ if one exists. Because the problem whether a group allows for an automatic presentation is not decidable, we cannot give a procedure that computes an automatic presentation if one exists or determines whether there is none.

Towards a feasible procedure we need to find a way to prove that an automatic presentation that resulted from a computation is correct. That is, given a finitely presented group $\mathfrak{G} := \mathrm{Mon}\langle A \mid R \rangle$ and automata $\mathfrak{W}$ and $(\mathfrak{M}_x)_{x\in A\cup\{\varepsilon\}}$, we want to verify that the automata do in fact form an automatic presentation for $\mathfrak{G}$. We need axioms, because we will need to do approximate computations of automatic presentations. Given the axioms in this section and enough memory and time, a very naive algorithm might just enumerate automatic presentations and check whether the current set of automata resembles an automatic presentation for the given group. This is probably a bad idea as it would take far too long.

In this section, we give a finite set of axioms dependent on $\mathrm{Mon}\langle A \mid R \rangle$ such that any tuple $\left(\mathfrak{W}, \mathfrak{M}_\varepsilon, (\mathfrak{M}_x)_{x\in A}\right)$ of automata is a model of these axioms if and only if they form an automatic presentation for the group presented by $\mathrm{Mon}\langle A \mid R \rangle$. Model checking is done by constructions involving finite state automata as demonstrated in Section 1.4. We restrict ourselves to **ShortLex** automatic presentations as this is about the only case for which a feasible algorithm to actually produce a word acceptor has been found.

Chapter 5 of [EPC$^+$92] gives a full set of thirteen axioms for general automatic presentations, so it would at least theoretically be possible to compute arbitrary automatic presentations for groups. A general procedure to compute automatic presentations for groups, based on the Todd-Coxeter enumeration scheme is presented there. The interested reader is thus referred to [EPC$^+$92] for more information on the general case. It might be interesting to know if any model-theoretic approach would yield new results about automatic presentations for groups.

We first present the axioms in a readable form. The class of models of these axioms contains, for a fixed group $\mathfrak{G}$ and a fixed **ShortLex** order on the language of strings over the generating set, all **ShortLex** automatic presentations for $\mathfrak{G}$. We note that the set of axioms is in fact finite and thus there are no problems that prevent us from model-checking a computed automatic presentation.

Definition 2.18 (axioms I):
Let $\mathfrak{G} = \mathrm{Mon}\langle A \mid R \rangle$ be a finitely presented group presented as a monoid and $\mathfrak{W}$ a finite state automaton over A and $\mathfrak{M}_x$ for each $x \in A$ finite state automata over $A^{\otimes 2}$. Consider the following axioms.

(1) *If $v \otimes w$ is accepted by one of the $\mathfrak{M}_x$ for $x \in A$, then v and w are also accepted by $\mathfrak{W}$.*

(2) *If $v \otimes w$ is accepted by one of the $\mathfrak{M}_x$ for $x \in A$, then $vx =_{\mathfrak{G}} w$.*

(3) *At least one string v in A^* is accepted by $\mathfrak{W}$.*

(4) *If for a string v in A^* and x in A, vx is accepted by $\mathfrak{W}$, then $v \otimes vx$ is accepted by $\mathfrak{M}_x$.*

(5) *Let (u, ε) be an element of R, that is u is an element of A^* and $u = x_0x_1\ldots x_n$. Then two strings w_0 and w_n are equal if and only if there are strings $w_1, \ldots, w_n$ accepted by $\mathfrak{W}$, such that $w_i \otimes w_{i+1}$ is accepted by $\mathfrak{M}_{x_i}$ for $1 \leqslant i \leqslant n$.* □

All the axioms given above have an equivalent FO$[\tau]$ formula over the signature $\tau = \left\{W, (M_x)_{x\in A\cup\{\varepsilon\}}\right\}$, where W is a predicate and the M_x are 2-ary relations. Thus by the method introduced in Section 1.4 we transform the conjunction of all axioms into a finite state automaton and check whether it accepts the empty tuple. For the sake of completeness we give $\mathrm{FO}[\tau]$ formulae over the signature $\tau = \left\{W, (M_x)_{x\in A\cup\{\varepsilon\}}\right\}$. The model-checking process takes up huge amounts of space and time. This is because of the existential constructions that come at the cost of an exponential blowup in state count. Thus a practical implementation needs to use efficient algorithms to be able to check larger examples.

Definition 2.19 (axioms II):
Let $\tau = \left\{W, M_\varepsilon, (M_x)_{x\in A}\right\}$ *be the signature for the following* $\mathrm{FO}[\tau]$ *axiom system.*

1) $\bigvee_{x\in A\cup\{\varepsilon\}} \forall v \forall w \left(M_x(v,w) \to W(v) \wedge W(w)\right)$

2) $\bigvee_{x\in A} \forall v \forall w \left(M_x(v,w) \to M_\varepsilon(vx,w)\right)$

3) $\exists v \left(W(v)\right)$

4) $\bigvee_{x\in A} \forall v \left(W(vx) \to M_x(v,vx)\right)$

5) $\bigwedge_{(x_1x_2\cdots x_n,\varepsilon)\in R} \forall w_0 \forall w_n \left(W(w_0) \wedge W(w_n) \to\right.$

$$\left.\left(w_0 = w_n \leftrightarrow \exists w_1 \ldots \exists w_{n-1} \left(\bigwedge_{1\leqslant i<n} W(w_i) \wedge \bigwedge_{1\leqslant i\leqslant n} M_{x_i}(w_i, w_{i+1})\right)\right)\right) \qquad \square$$

The following theorem ensures that if we found the axioms to hold for a finitely presented group $\mathfrak{G} = \mathrm{Mon}\langle A \mid R \rangle$ and tuple of automata we found an automatic presentation for $\mathfrak{G}$.

Theorem 2.20 (correctness of axioms):
Let $\mathfrak{G} := \mathrm{Mon}\langle A \mid R \rangle$ *be a group presented as a monoid and let* $\mathfrak{W}$ *be a finite state automaton over* A *and* $\mathfrak{M}_x$ *for* $x \in A\cup\{\varepsilon\}$ *be finite state automata over* $A^{\otimes 2}$. *Then* $\mathfrak{W}$ *and the* $\mathfrak{M}_x$ *for* $x \in A\cup\{\varepsilon\}$ *form a prefix-closed automatic presentation with uniqueness for* $\mathfrak{G}$ *if and only if the axioms in Definition 2.18 hold.*

Proof: Let $\mathfrak{a} = \left(A, \pi, \mathfrak{W}, \mathfrak{M}_\varepsilon, (\mathfrak{M}_a)_{a\in A}\right)$ be a prefix-closed automatic presentation with uniqueness for the finitely presented group $\mathfrak{G}$. We show that this structure is a model of the axioms. By Definition 2.1, for any string $v\otimes w$ that is an element of $L(\mathfrak{M}_x)$ for $x \in A\cup\{\varepsilon\}$, v as well as w are also elements of $L(\mathfrak{W})$ and also by this definition, if $v\otimes w$ is an element of one of the $L(\mathfrak{M}_x)$, then $vx =_{\mathfrak{G}} w$, thus 1) and 2) hold. Because π has to be surjective, $L(\mathfrak{W})$ cannot be empty, thus 3) is true. As $L(\mathfrak{W})$ is prefix closed v is an element of $L(\mathfrak{W})$ if vx is an element of $L(\mathfrak{W})$ and thus $\mathfrak{M}_x$ accepts $v\otimes vx$, which tells us that 4) holds.

To show that 5) holds, we assume $\pi(w_0x_1\cdots x_i)$ to have a unique representative v_i in $L(\mathfrak{W})$. Since $x_1x_2\cdots x_n$ is equal to the identity in $\mathfrak{G}$, w_0 and w_n represent the same element of $\mathfrak{G}$ and are thus equal because of the uniqueness property of $L(\mathfrak{W})$. Futhermore $v_{i-1}\otimes v_i$ is accepted by $\mathfrak{M}_{x_i}$ because $v_{i-1}x_i =_{\mathfrak{G}} v_i$. Let $w_0 = w_n$. Then we can take v_i as w_i for $1 \leqslant i < n$. Conversely assume that there are strings w_i for $1 \leqslant i < n$ in $L(\mathfrak{W})$ and such that $w_{i-1}\otimes w_i$ is accepted by $\mathfrak{M}_{x_i}$. Then the w_i already are shortest representatives and $w_n = w_0$. This takes care of 5).

Conversely, let axioms 1) to 5) hold for $\mathfrak{G} = \mathrm{Mon}\langle A \mid R \rangle$ and an automatic presentation $\mathfrak{a}$. We have to show that $\mathfrak{a}$ is in fact an automatic presentation for $\mathfrak{G}$. We will accomplish this by showing that the automatic presentation defines a group that is isomorphic to $\mathfrak{G}$ and that Definition 2.1 is fulfilled.

Also, we have to show that the word acceptor has a prefix closed language and contains a unique representative for each group element.

We first show prefix-closedness of $L(\mathfrak{W})$. Following from 4), if vx is an element of $L(\mathfrak{W})$, then v also is an element of $L(\mathfrak{W})$, thus $L(\mathfrak{W})$ is prefix closed. In particular ε is contained in $L(\mathfrak{W})$ by 3).

We now want to look into how the automatic presentation defines a group. For this, we first want to define maps

$$\sigma_x : L(\mathfrak{W}) \to L(\mathfrak{W}),$$

that map every element v in $L(\mathfrak{W})$ to a representative of $\pi(vx)$ in $L(\mathfrak{W})$. We make ourselves clear, why these maps are well-defined and bijective. Let v be in $L(\mathfrak{W})$ and x be an element of A. There is by hypothesis an element (xX, ε) in R. We want to apply 5) and take $w_0 = w_2 = v$. By 5) there exists a string w in $L(\mathfrak{W})$ with $v \otimes w$ in $L(\mathfrak{M}_x)$, thus the image of v is defined. Also $w \otimes v$ in $L(\mathfrak{M}_X)$. For well-definedness let w' be in $L(\mathfrak{W})$ with $v \otimes w'$ in $L(\mathfrak{M}_x)$. We know that $w \otimes v$ is in $L(\mathfrak{M}_X)$ and that (Xx, ε) is an element of R. Thus, again by 5), w equals w' and thus σ_x is both well-defined and bijective.

We define the map

$$\varphi : A \to \mathfrak{S}_{L(\mathfrak{W})} : x \mapsto (v \mapsto \sigma_x(v)),$$

where $\mathfrak{S}_{L(\mathfrak{W})}$ denotes the symmetric group of the set $L(\mathfrak{W})$. The map φ extends to a group homomorphism from $\mathcal{F}(A)$ to $\mathfrak{S}_{L(\mathfrak{W})}$. Because this map respects the relations in the presentation of $\mathfrak{G}$, it defines a faithful action of $\mathfrak{G}$ on $L(\mathfrak{W})$ and the group defined by the automatic presentation is a monomorphic image of $\mathfrak{G}$ in $\mathfrak{S}_{L(W)}$.

Now let $g \in G$ be equal to $a_1 \cdots a_n$ with a_i in A for $1 \leqslant i \leqslant n$. Then $\varphi(g) = \varphi(a_1) \cdots \varphi(a_n)$ because φ is a group homomorphism from $\operatorname{Mon}\langle A \mid R \rangle$ to $\mathfrak{S}_{L(\mathfrak{W})}$ and the image of ε under the permutation $\varphi(a_1) \cdots \varphi(a_n)$ also represents g because of 2). Because A is a generating set for $\mathfrak{G}$, the language $L(\mathfrak{W})$ contains a representative for every group element of $\mathfrak{G}$ and π is thus surjective, as the definition of an automatic presentation demands. In fact

$$\tau : G \to L(\mathfrak{W}) : g \mapsto \varphi(g)(\varepsilon)$$

is a right inverse for π.

Now let g also be represented by $b_1 \cdots b_m$. Because φ is a homomorphism

$$\varphi(a_1) \cdots \varphi(a_n) = \varphi(a_1 \cdots a_n) = \varphi(g) = \varphi(b_1 \cdots b_m) = \varphi(b_1) \cdots \varphi(b_m).$$

By 2), $\varphi(a_1) \cdots \varphi(a_n)$ maps ε to $a_1 \cdots a_n$ and $\varphi(b_1) \cdots \varphi(b_m)$ maps ε to $b_1 \cdots b_m$. But now by we have $a_1 \cdots a_n = b_1 \cdots b_m$, thus $\mathfrak{W}$ is a unique word acceptor and τ is also a left inverse for π.

As last step, we check that for $x \in A \cup \{\varepsilon\}$ the string $v \otimes w$ is accepted by $\mathfrak{M}_x$ if and only if $vx =_{\mathfrak{G}} w$. Let for this $v \otimes w$ be an element of $L(\mathfrak{M}_x)$ for some x in $A \cup \{\varepsilon\}$. Then by 1) the strings v as well as w are elements of $L(\mathfrak{W})$ and by 2) the equality $\pi(vx) = \pi(w)$ holds. Let conversely v and be an element of $L(\mathfrak{W})$ and $x \in A \cup \{\varepsilon\}$. Then there is a unique element w such that $v \otimes w$ is accepted by $\mathfrak{M}_x$ and $vx =_{\mathfrak{G}} w$, as shown above.

■

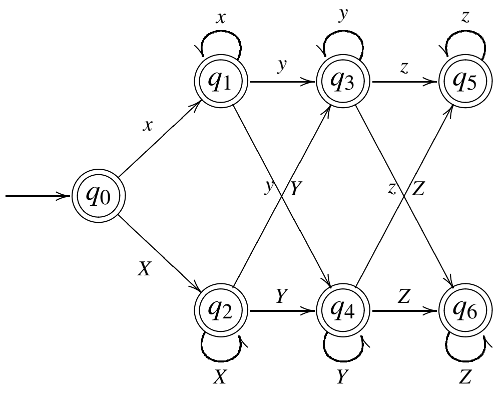
Thus we have a constructive and algorithmically feasible means to check whether an automatic presentation that resulted from a computation is in fact an automatic presentation for the given group. This is a crucial point when trying to solve semi-decidable problems.

2.4 Limitations of Automatic Presentations

We shortly look at a basic example to show limitations of the concept of automatic presentations in the sense of Section 2.1. In fact automatic groups are a large but restricted class of finitely presented groups.

Let for this section $A := \{x, X, y, Y, z, Z\}$, where the capital letters denote the formal inverses of the generators in small letters. We define a finite state automaton $\mathfrak{W}$ with input alphabet A as follows.

Definition 2.21 (a regular language):



□

It is easy to see that $\mathfrak{W}$ accepts strings that start with a number of either xs or Xs followed by a number of either ys or Ys and then a number of zs or Zs.

Consider the following two non-isomorphic groups. The first one is the free abelian group on three generators, the notation $[x,y]$ is a short form for the commutator of x and y, that is $[x,y] = xyXY$. Putting a commutator into the set of relations makes the involved generators commute in the presented group.

$$\mathfrak{G} := \mathrm{Mon}\langle\, x, X, y, Y, z, Z \mid [x,y], [x,z], [y,z] \,\rangle .$$

The second one is a so-called *Heisenberg group*, which is also finitely generated by three generators, but has slightly different relations.

$$\mathfrak{H} := \mathrm{Mon}\langle\, x, X, y, Y, z, Z \mid [x,y]Z, [x,z], [y,z] \,\rangle .$$

It is easily seen that $\mathfrak{G}$ and $\mathfrak{H}$ are not isomorphic.

We go on to show that there are surjective maps $\pi : L(\mathfrak{W}) \to G$ as well as $\pi' : L(\mathfrak{W}) \to H$. For this we rewrite any string over the generators in A into a normal form. In the case of the free abelian group on three generators this is easy. Given an arbitrary string over A we can move all x and X to the front, followed by all y and Y and then z and Z.

For the Heisenberg group this is only slightly more complex. Everytime we exchange a y with an x we get an additional z, but it is also possible to form a string in $L(\mathfrak{W})$.

Thus, the language of a word acceptor does not in any way determine the group we are talking about. This gets worse, because $\mathfrak{H}$ is not automatic. It is however possible to find an automatic presentation for the Cayley graph of $\mathfrak{H}$ in the sense of Definition 1.50 this is shown in [BG04].

The article [OT05] characterises finitely generated automatic groups as the class of groups that have an abelian subgroup of finite index. This is a generalisation of a result in [EPC^{+}92] that states that groups that are virtually abelian are automatic.

3 Algorithms for Automatic Presentations

"Beware of bugs in the above code; I have only proved it correct, not tried it."

(Donald Knuth)

After we promised in the previous chapter that automatic groups allow for effective algorithms, we now have to back up our claims. We first introduce an algorithm that solves the word problem for automatic groups in time quadratic in the length of the input string. The fact that the word problem for automatic groups is decidable in quadratic time already gives a hint that automatic groups are a rather restricted class of groups. We then turn to the question whether we can actually compute an automatic presentation for a finitely presented group. This is in fact one of the most important results of this theory. We cannot exploit the algorithmic properties of automatic groups if we are not able to construct automatic presentations. Unfortunately, to be able to compute automatic presentations we have to restrict ourselves again, this time to the class of **ShortLex** automatic groups. It is known that there are automatic groups that do not have a **ShortLex** automatic presentation. An example of such a group is given in [EPC+92], Chapter 3.

The most we can hope for is an algorithm that finds an automatic structure if one exists and does not terminate if not. In other words: the question whether an automatic presentation exists is semi-decidable. We want to be a bit more efficient than just enumerating automata and towards a feasible procedure we use the well-known Knuth-Bendix completion procedure to be able to construct a word difference automaton and reduce strings to a normal form.

3.1 Word Problem

In this section we show how to solve the word problem in an automatically presented group. We recall that the word problem for a group presentation is the question whether two strings in the generators represent the same group element, or equivalently, whether a string over the generators represents the identity element of the group. The presented algorithm has quadratic runtime with respect to the length of the input and is essentially an existential construction in the multiplication automata.

In a preparatory step we find for a given string v in $L(\mathfrak{W})$ and an element x of A, a string w that also is an element of $L(\mathfrak{W})$ such that $v \otimes w$ is accepted by $\mathfrak{M}_x$. This means we find a representative for $\pi(v)x$ in $L(\mathfrak{W})$. Effectively we have also found an algorithm to multiply group elements.

When given $R = \{(q_0, \varepsilon)\}$ and v as input, Algorithm 1 performs a breadth-first search inside the multiplication automaton. Remember that the input alphabet of the multiplication automata is $A^{\otimes 2}$, thus every transition is of the form $\left(p, \begin{bmatrix} x \\ y \end{bmatrix}, q\right)$. The algorithm finds all runs of $\mathfrak{M}_x$ that have the string v possibly followed by padding symbols in the first component of the input. If the algorithm reaches an accepting state at some point, it outputs the label in the second component. This construction is similar to the one given in Lemma 1.47 and the construction of the map η for existential quantification.

Correctness of the algorithm is clear by the definition of an automatic presentation for groups. The string $v \otimes w$ is accepted by $\mathfrak{M}_x$ thus w has to be an element of $\mathfrak{W}$ and $vx =_{\mathfrak{G}} w$. We want to guarantee that the computaion terminates. First, the algorithm has to consume the whole of v but then there might be a number of padding symbols needed. Because of Lemma 2.15 there can only be a bounded number of blank symbols. Thus the output string is of length bounded by $|v| + N$.

The error returned in Line 2 should never happen, right multiplication by a generator is always defined in a group and therefore we have to be able to find a representative for vx in $L(\mathfrak{W})$.

We briefly analyse the runtime of the algorithm. Because v has finite length, Algorithm 1 calls itself $|v| + N$ times where N is a constant that depends on the automatic presentation. In every step there is a bounded amount of elements that are contained in the set R because $\mathfrak{M}_x$ only has finitely many states and every state has only finitely many outgoing transitions. The runtime is in $O(|v|)$.

Algorithm 1: multiplyByGenerator()

```
Input: A set $R$ of pairs $(q,s)$ where $q$ is a state and $s$ is a string over $A$,
an automaton $\mathfrak{M}_x$,
a string $v$ in $L(\mathfrak{W})$
Output: A string $w$ in $L(\mathfrak{W})$ that represents $vx$.
1  begin
2      if $R = \emptyset$ then return error
3      if $v = \varepsilon$ and there is $(q,s)$ in $R$ with $q$ in $F$ then
4          return $s$
5      else
6          if $|v| > 0$ then $v' \longleftarrow v[1]$ else $v' \longleftarrow \square$
7          $R' \longleftarrow \left\{ (q,sy) \mid \left(p, \begin{bmatrix} v' \\ y \end{bmatrix}, q\right) \in \tau \text{ and } (p,s) \in R \right\}$
8          multiplyByGenerator($R'$, $\mathfrak{M}_x$, $v_2 \cdots v_n$)
9      end
10 end
```

We give an algrithm that solves the word problem for automatic groups in quadratic time in the length of the input string.

Theorem 3.1 (word problem in quadratic time):
Let $\mathfrak{G}$ be an automatic group. Then for any string v over A we can compute a string w in $L(\mathfrak{W})$ in time proportional to $|v|^2$ representing the same element as v in $\mathfrak{G}$.

The word problem can be solved by computing for a given string w a representative w' in $L(\mathfrak{W})$ and then comparing w' to a representative of the identity using $\mathfrak{M}_\varepsilon$.

Proof: We use Algorithm 1. Starting with a representative of length n_0 for the identity element we find a representative for $v[i]$ by multiplying a representative for $v[i-1]$ with the generator at the i-th position of v. In every step the length of the output is bounded by $|v[i]| + N$, and thus the runtime of the algorithm that solves the word problem is bounded by

$$\sum_{i=1}^{|v|} iN + n_0,$$

and thus the runtime of this procedure is in $O\left(|v|^2\right)$.

To show that we can determine a representative e of the identity in $\mathfrak{G}$, we determine an accepted string in $L(\mathfrak{W})$ and multiply it sucessively with its formal inverse. That is with e reversed and every generator replaced by its formal inverse.

■

3.2 String Rewriting and Knuth-Bendix Completion

Before we proceed to the algorithm that computes an automatic presentation, we take a short excursion to the theory of string rewriting systems. String rewriting systems play a central role in many algorithms involving finitely presented groups and finitely presented algebras. It will become clear shortly why this is the case. A very thorough treatment of rewriting systems for finitely presented groups can be found in [Sim94]. We will only give the theory as it is needed for our purposes.

We recall the definition of a finitely presented monoid $\mathfrak{M} = \mathrm{Mon}\langle A \mid R \rangle$. The monoid $\mathfrak{M}$ is a set of congruence classes and we would like to have a transversal for the congruence classes, but it is not at all clear how to choose such a transversal. To define which elements we prefer over others, we employ the idea of an order on the set of strings over A. For our transversal to be of any value, we wish to be able to to choose a unique string as representative. Additionally, we want to be able to compute this representative when given an arbitrary string over A. We employ the concept of a well-order on A^*.

In the context of finitely presented monoids, we might first choose a linear ordering of the set A of generators. Because A is finite, there is a well defined number of linear orders on A: a set of n elements has $n!$ distinct linear orders, all of which are also well-orders. Unfortunately, there is no unique way to extend a linear order on A to a linear order on A^*, much less a well-order.

To extend an order on A to a well-order on A^* we choose the following well order. Let $v = v_1 \cdots v_n$ and $w = w_1 \cdots w_m$ be two strings over A. Then $v \leqslant_{\mathrm{SL}} w$ in the **ShortLex** order on A^* if either $n < m$ or $n = m$ and there is a k with $1 \leqslant k \leqslant n$, such that $v_i = w_i$ for $1 \leqslant i < k$ and $v_k \leqslant_A w_k$. For an alphabet of cardinality n we have thus defined $n!$ distinct **ShortLex** orders on A^*. There are further ways to define well-orders on the free monoid, but we will not deal with them at this point.

For the rest of this section we can choose any well-order on A^* we like and $\leqslant$ always denotes a well-order. We have another property we want the well-order to have, called translation invariance.

Definition 3.2 (translation invariance and reduction order):
Let A be an alphabet and let $\leqslant$ be an order on A^*. Then $\leqslant$ is called translation invariant, *if for strings v and w in A^* with $v \leqslant w$ we have $avb \leqslant awb$ for all strings a and b in A^*.*
We call a translation invariant well-order on A^* a reduction order. □

We note that the **ShortLex** order defined above is translation invariant.

Now every subset of A^* has a unique minimal element with respect to $\leqslant$. Thus if we have a congruence on A^* every congruence class contains a unique minimal element. At this point finding this element is not constructive. We do not have a way to tell which strings belong to the same congruence class algorithmically. We will be optimistic and assume that we actually can find a shortest string in the congruence class of a given string at least for some cases. For this we need the notion of a rewriting system. The idea behind this concept is that we rewrite strings over an alphabet using a set of well defined rules. Strings that can be rewritten to each other using these rules are considered equal under the rewriting system.

Definition 3.3 (string rewriting system):
Let A be an alphabet. A string rewriting system *or* RWS, *is a set $\mathcal{S}$ of pairs of strings, called* rules. *A rule is usually denoted $v \to w$, indicating that a string v can be rewritten to w. In this context v is the* left-hand side *of the rule and w is the* right-hand side *of the rule.*

If $\leqslant$ is a reduction order on we normally want $w \leqslant v$ to hold for all rules $v \to w$ in $\mathcal{S}$. □

Given a rewriting system $\mathcal{S}$, let $v \to w$ be a rule in $\mathcal{S}$. We say that the rule $v \to w$ *matches* a string s if $s = xvy$ with x and y in A^*. The string s can be rewritten to xwy using $\mathcal{S}$. Of course there can be many rules matching a given string s. A string s over A is *irreducible* if no rule matches it. The set of strings that are irreducible under $\mathcal{S}$ is denoted $\mathrm{Irr}_{\mathcal{S}}$. Usually we want to apply rewrite rules as long as we can until we reach an irreducible string. We call a sequence $s \to s' \to \ldots \to t$, where in each step one rewrite rule is applied, a chain of rewrite steps and call irreducible strings t that result from rewriting some string s using the rewriting system a *residue* of s.

Let $\mathfrak{M} := \mathrm{Mon}\langle A \mid R \rangle$ be a finitely presented monoid. Recall that R is a finite set of pairs of strings that generates a congruence relation $\sim_R$ on $\mathcal{M}(A)$. Without loss of generality we can assume $w \leqslant v$ for all pairs (v,w) in R and thus make R into a rewriting system.

We extend the rewriting relation to its transitive closure. We let $v \to^* w$, if there are strings $v = u_1, \ldots, u_n = w$ with $u_i \to u_{i+1}$. Reflexivity is gained by adding the trivial rewrite rule $\varepsilon \to \varepsilon$. To form an equivalence relation we also need symmetry, thus we denote by $\leftrightarrow^*$ the symmetric closure of $\to$ and let $v \leftrightarrow^* w$ if and only if there are strings $v = u_0, u_1, u_2, \ldots, u_n = w$ with

$$\begin{tikzcd}
& u_1 \arrow[dl, "*"'] \arrow[dr, "*"] & & u_3 \arrow[dl, "*"'] \arrow[dr, "*"] & & u_{n-1} \arrow[dl, "*"'] \arrow[dr, "*"] & \\
u_0 & & u_2 & & \cdots & & u_n.
\end{tikzcd}$$

The two congruence relations $\sim_{\mathcal{S}}$ and $\leftrightarrow^*$ coincide. A string that is the unique least element in its $\leftrightarrow^*$ congruence class is called *$\leftrightarrow^*$-minimal* or *$\mathcal{S}$-minimal*.

We want to ensure that we can actually compute for a given string v a unique representative of its congruence class. For this it is important that rewriting stops after a finite amount of time and that it does not matter in what order we apply rewrite rules. A rewriting system $\mathcal{S}$ is called *Noetherian*, if every chain of rewrite steps reaches an irreducible string after a finite number of steps. A rewriting system is *confluent*, if for every string v that can be rewritten in two distinct ways, $v \to^* u_1$ and $v \to^* u_2$, there is a string w such that $u_1 \to^* w$ and $u_2 \to^* w$.

Local confluence is a slightly weaker property of rewriting systems, meaning that for every string v that can be rewritten in two distict ways, $v \to u_1$ and $v \to u_2$ in just one rewriting step, there exists a string w such that $u_1 \to^* w$ and $u_2 \to^* w$.

Confluence ensures that no matter in what order we apply rewrite rules, we always reach the same irreducible string.

We call a rewriting system *complete* if it is Noetherian and confluent. If we choose a reduction order, then $\mathcal{S}$ is Noetherian, because if it were not an infinite reduction chain would be a contradiction to $\leqslant$ being a well order.

A complete rewriting system enables us to rewrite a string to a unique normal form. We will reach an irreducible string after a finite number of rewriting steps. Confluence guarantees that we reach a unique string regardless of the order in which we apply the rewriting rules. Thus completeness seems to be a desireable property for rewriting systems to have, and the soon to be introduced Knuth-Bendix completion process enables us to complete any given set of rewrite rules.

For Noetherian rewriting systems it suffices to have local confluence for completeness, as local confluence and confluence are equivalent in this case. This is not be true for rewriting systems that are not Noetherian.

Lemma 3.4 (Noetherian and locally confluent):
Let $\mathcal{S}$ be a Noetherian and locally confluent rewriting system. Then $\mathcal{S}$ is complete.

Proof: We show that every string s has a unique residue t with $s \rightarrow^* t$. This implies confluence.

As $\mathcal{S}$ is Noetherian, every sequence of rewriting steps is finite. Let $s = u_0 \rightarrow u_1 \rightarrow u_2 \rightarrow \cdots \rightarrow u_k$ with u_k irreducible be a rewrite chain of minimal length and $s = u_0 \rightarrow u'_1 \rightarrow u'_2 \rightarrow \cdots \rightarrow u'_m$ be an arbitrary rewrite chain with u'_m irreducible.

We use induction on the length of a shortest rewriting chain. For $k = 0$ this is clear, because s is irreducible in that case.

Let now k be greater than zero. By local confluence, there is a string v such that $u_1 \rightarrow^* v$ and $u'_1 \rightarrow^* v$. Let $v \rightarrow^* w$ with w irreducible. By induction, $w = u_k$ and thus $w = u'_m$ and thus $u_k = u'_m$.

Thus, any reduction chain for s has to yield the same irreducible string. This implies the existence of an unique irreducible for each string in A^* under $\mathcal{S}$ and thus confluence.

■

From Lemma 3.4 we deduce that every congruence class of strings under a Noetherian and locally confluent rewriting system has a unique irreducible representative. In the case of the **ShortLex** order, this is the minimal string in the equivalence class in the **ShortLex** order.

Corollary 3.5 (unique representatives):
For a complete rewriting system $\mathcal{S}$ every equivalence class contains a unique irreducible representative.

Proof: Let $\mathcal{S}$ be a complete rewriting system and u and v be irreducible strings with $u \leftrightarrow^* v$. We have to show that u is in fact equal to v. Because $u \leftrightarrow^* v$, there exist strings u_i for $0 \leqslant i \leqslant n$ with $u_0 = u$ and $u_n = v$, such that either $u_i \rightarrow^* u_{i+1}$ or $u_{i+1} \rightarrow^* u_i$. We want to show that $u = v$ and use induction on n.

The result is clear for $n = 0$, because u equals v in this case. If $n > 0$ there exists a string s such that $u \rightarrow^* s$ and $u_2 \rightarrow^* s$ by confluence of $\mathcal{S}$. Because u is irreducible, u_2 is equal to u. By induction $u_n = u$ and thus $u = v$. ■

We characterise local confluence by conditions that only rely on the set of rewrite rules. This will enable us to check algorithmically whether a set of rewrite rules is locally confluent. For this it is useful to look at the cases where local confluence fails to hold.

Lemma 3.6 (testing local confluence):
Let A be an alphabet and $\mathcal{S}$ be a rewriting system. Then $\mathcal{S}$ is locally confluent if and only if the following conditions hold for any two rules $v_1 \rightarrow w_1$ and $v_2 \rightarrow w_2$ in $\mathcal{S}$.

(i) If $v_1 = xy$ and $v_2 = yz$ for x, y, z in A^* and $y \neq \varepsilon$, then $w_1 z \rightarrow^* t$ and $x w_2 \rightarrow^* t$ for some t in A^*.

(ii) If $v_1 = x w_2 y$ for x, y in A^* then $w_1 \rightarrow^* w$ and $x w_2 y \rightarrow^* w$ for some w in A^*.

Proof: If $\mathcal{S}$ is locally confluent the above conditions hold by definition of local confluence.

Let us now assume that (i) and (ii) hold for all pairs of rules in $\mathcal{S}$. We choose a string s from A^* such that we can apply two rules $v_1 \rightarrow w_1$ and $v_2 \rightarrow w_2$. If s is of the form $x v_1 y v_2 z$ we can rewrite it in the obvious two ways to $t_1 = x w_1 y v_2 z$ and $t_2 = x v_2 y w_2 z$. These two strings obviously can be rewritten to $w = x w_1 y w_2 z$ using the respective rewrite rules.

If, however, there is an overlap of v_1 and v_2 in s we have one of the situations given in (i) or (ii) and can guarantee the confluence condition. ■

For finite sets of rewrite rules, the above lemma yields an algorithmically feasible test for confluence, and for infinite sets we can at least check for local confluence on strings that are shorter than some n. We take all pairs $v_1 \to w_1$ and $v_2 \to w_2$ of rules in $\mathcal{S}$ in turn and check for overlaps in the left hand sides v_1 and v_2. If we find such an overlap we fuse the two left hand sides together and do two distinct possible rewrite steps. We then rewrite the resulting strings using $\mathcal{S}$ and if the residues are equal the test succeeds. Lemma 3.6 above also tells us what to do in case we get two different residues u_1 and u_2: just add a new rule $u_1 \to u_2$, assuming $u_2 \leqslant u_1$ in the chosen reduction order.

For some arguments as well as practical implementations, we have to take care of an ambiguity that turns up when choosing the order of reductions to apply. As the rewriting system may not yet be complete, we may get different irreducibles when applying rewrite rules in different orders. Thus, we define a canonical reduction. Our choice of canonical reduction is arbitrary. There are quite a few other possible choices.

Definition 3.7 (canonical reduction):
Let $\mathcal{S}$ be a rewriting system. We define the canonical reduction *$\rho_{\mathcal{S}}(s)$ of a string s in A^*. From all rules in $\mathcal{S}$ that can be applied to s, we choose the rules that can be applied as far to the left as possible. Among these rules we choose the rules with the shortest left hand side. Among the remainung rules we choose the one with the least right hand side with respect to the chosen reduction reduction order. We apply this rule and repeat the process until we reach an irreducible string. We call this string the canonical reduction of s, denoted $\rho_{\mathcal{S}}(s)$.* □

We might end up adding more and more rules to the set of rewrite rules and unfortunately in most cases a complete system of rules is infinite. We cannot hope to tell in general whether we find a finite complete set of rewrite rules, because then we could decide the word problem for monoids.

In the process of adding new rules we might generate rules that obsolete older ones. The following lemma allows us to delete certain rules to reduce the number of rewrite rules in a rewriting system. This mainly is important for practical purposes, because the algorithms involved in Knuth-Bendix completion and string rewriting have to match all possible rules to a string, and if we can avoid work this can only be good.

We call a rewriting system $\mathcal{S}$ *reduced*, if for all $v \to w$ in $\mathcal{S}$, the strings v as well as w are irreducible under $\mathcal{S} \setminus \{v \to w\}$. We want to take a short look at how we can obtain a reduced set of rewrite rules from an arbitrary one. We have to ensure that the relation $\to^*$ stays unaltered.

Lemma 3.8 (reducing rewriting systems):
Let $\mathcal{S}$ be a rewriting system, $v \to w$ be a rule in $\mathcal{S}$ and $\mathcal{S}'$ the rewriting system $\mathcal{S}$ without the rule $v \to w$. We distinguish the following three cases.

- *If $\rho_{\mathcal{S}'}(v) = \rho_{\mathcal{S}'}(w)$, we do not add anything to $\mathcal{S}'$ and just set $\mathcal{S}''$ equal to $\mathcal{S}'$.*
- *If $\rho_{\mathcal{S}'}(v) > \rho_{\mathcal{S}'}(w)$ we add the rule $\rho_{\mathcal{S}'}(v) \to \rho_{\mathcal{S}'}(w)$ to $\mathcal{S}'$, calling the new rewriting system $\mathcal{S}''$.*
- *If $\rho_{\mathcal{S}'}(w) > \rho_{\mathcal{S}'}(v)$ we add the rule $\rho_{\mathcal{S}'}(w) \to \rho_{\mathcal{S}'}(v)$, again yielding $\mathcal{S}''$.*

Then $\to^*_{\mathcal{S}}$ equals $\to^*_{\mathcal{S}''}$. The rewriting system $\mathcal{S}$ is complete if and only if $\mathcal{S}''$ is complete.

Proof: The three operations named above certainly do not change the congruence generated by $\mathcal{S}$, because we only replace strings by strings equivalent under $\leftrightarrow^*$.

We show that $s \to^*_{\mathcal{S}} t$ if and only if $s \to^*_{\mathcal{S}''} t$ in all of the three cases. For this assume $v \to w$ to be the rule subject to minimisation. We assume without loss of generality the rule $v \to w$ to be applied only in the first rewrite step.

The canonical $\mathcal{S}$ reduction of s looks like this.

$$s = xvy \to xwy \to^*_{\mathcal{S}'} x\rho_{\mathcal{S}'}(w)y \to^*_{\mathcal{S}'} t$$

We look at what happens in the three cases. If $\rho_{\mathcal{S}'}(v) = \rho_{\mathcal{S}'}(w)$, then the reduction is as follows.

$$s = xvy \to^*_{\mathcal{S}''} x\rho_{\mathcal{S}'}(v)y = x\rho_{\mathcal{S}'}(w)y \to^*_{\mathcal{S}''} t$$

If $\rho_{\mathcal{S}'}(v) > \rho_{\mathcal{S}'}(w)$, then

$$s = xvy \to^*_{\mathcal{S}''} x\rho_{\mathcal{S}'}(v)y \to x\rho_{\mathcal{S}'}(w)y \to^*_{\mathcal{S}''} t$$

The last case is a bit special, because $\rho_{\mathcal{S}'}(v)$ is smaller in the reduction order than $\rho_{\mathcal{S}'}(w)$, there is a reduction under $\mathcal{S}'$ that does not need the rule $v \to w$ to be applied to reach t. We first look at the reduction under $\mathcal{S}$ under these circumstances:

$$s = xvy \to xwy \to^*_{\mathcal{S}'} x\rho_{\mathcal{S}'}(w)y \to x\rho\mathcal{S}'(v)y \to^*_{\mathcal{S}'} t$$

And now it is fairly clear, that if we replace $v \to w$ by $\rho_{\mathcal{S}'}(w) \to \rho_{\mathcal{S}'}(v)$ this does not influence the reduction, because

$$s = xvy \to^*_{\mathcal{S}'} x\rho_{\mathcal{S}'}(v)y \to^*_{\mathcal{S}'} t.$$

Because we do not change the residues of any string, $\mathcal{S}$ is complete if and only $\mathcal{S}''$ is complete.

■

We give the Knuth-Bendix procedure as described above as pseudo-code. There is a large body of literature that deals with how to implement the rewriting process efficiently, but we will not go into further detail here.

First, this short example shows what the Knuth-Bendix completion process does for string rewriting systems.

Example 3.9 (Knuth-Bendix completion process):
Let $v_1 \to w_1$ and $v_2 \to w_2$ be two rules in a rewriting system $\mathcal{S}$. Let $v_1 = xy$ and $v_2 = yz$ for strings x, y and z with y not equal to ε. Then the Knuth-Bendix procedure does the following.

$$\begin{tikzcd}
 & w_1x \arrow[r, "*"] & u_1 \\
\underline{\underline{xyz}} \arrow[ur] \arrow[dr] & & \\
 & xw_2 \arrow[r, "*"] & u_2
\end{tikzcd}$$

If t_1 is equal t_2, then local confluence holds for the two rules. If t_1 is not equal to t_2 the procedure adds a new rule $t_1 \to t_2$ or $t_2 \to t_1$ depending on whether $t_2 \leqslant t_1$ or $t_1 \leqslant t_2$ in the chosen reducion order.□

For a given rewriting system $\mathcal{S}$ Algorithm 2 performs one step of Knuth-Bendix completion. As described in Lemma 3.8 we simplify a set of rules using Algorithm 3. In conclusion Algorithm 4 gives a procedure that tries to find a complete set of rewrite rules and outputs a sequence of sets of rules. Algorithm 4 is only guaranteed to terminate if there is a finite completion for $\mathcal{S}$.

Algorithm 2: KnuthBendixCompletion()

```
Input: A set $\mathcal{S}$ of rewrite rules and a reduction order $\leqslant$.
Output: A set $\mathcal{S}'$ of rewrite rules.
begin
    $\mathcal{S}' \longleftarrow \mathcal{S}$
    foreach pair $v_1 \to w_1$ and $v_2 \to w_2$ of rules in $\mathcal{S}$ do
        if there are strings $x$, $y$ and $z$ such that $y \neq \varepsilon$ and $v_1 = xy$ and $v_2 = yz$ then
            $t_1 \longleftarrow \rho_{\mathcal{S}}(xw_2)$
            $t_2 \longleftarrow \rho_{\mathcal{S}}(w_1z)$
        else if there are strings $x$ and $y$ such that $v_1 = xv_2y$ then
            $t_1 \longleftarrow \rho_{\mathcal{S}}(w_1)$
            $t_2 \longleftarrow \rho_{\mathcal{S}}(xw_2y)$
        end
        if $t_1 \neq t_2$ then
            if $t_1 \leqslant t_2$ then
                $\mathcal{S}' \longleftarrow \mathcal{S}' \cup \{t_2 \to t_1\}$
            else if $t_2 \leqslant t_1$ then
                $\mathcal{S}' \longleftarrow \mathcal{S}' \cup \{t_1 \to t_2\}$
            end
        end
    end
    return $\mathcal{S}'$
end
```

Algorithm 3: SimplifyRules()

```
Input: A set $\mathcal{S}$ of rewrite rules and a reduction order $\leqslant$.
Output: A reduced set $\mathcal{S}$ of rewrite rules.
begin
    foreach rule $v \to w$ in $\mathcal{S}$ do
        $\mathcal{S}' \longleftarrow \mathcal{S} \setminus \{v \to w\}$
        if $\rho_{\mathcal{S}'}(v) = \rho_{\mathcal{S}'}(w)$ then
            $\mathcal{S} \longleftarrow \mathcal{S}'$
        else if $\rho_{\mathcal{S}'}(v) \leqslant \rho_{\mathcal{S}'}(w)$ then
            $\mathcal{S} \longleftarrow \mathcal{S}' \cup \{\rho_{\mathcal{S}'}(w) \to \rho_{\mathcal{S}'}(v)\}$
        else if $\rho_{\mathcal{S}'}(v) \leqslant \rho_{\mathcal{S}'}(w)$ then
            $\mathcal{S} \longleftarrow \mathcal{S}' \cup \{\rho_{\mathcal{S}'}(v) \to \rho_{\mathcal{S}'}(w)\}$
        end
    end
    return $\mathcal{S}$
end
```

Algorithm 4: KnuthBendix2()

```
Input: A set $\mathcal{S}$ of rewrite rules, a reduction order $\leqslant$ and an interruption parameter $k \in \mathbb{N}$
Output: A possibly non-terminating sequence $\mathcal{S}_i$ of reduced sets rewrite rules.
begin
    $i \longleftarrow 0$
    $\mathcal{S}_0 \longleftarrow \mathcal{S}$
    repeat
        $\mathcal{S}_i' \longleftarrow$ SimplifyRules($\mathcal{S}_i, \leqslant$)
        $\mathcal{S}_i'' \longleftarrow$ KnuthBendixCompletion($\mathcal{S}_i', \leqslant$)
        $\mathcal{S}_{i+1} \longleftarrow \mathcal{S}_i''$
        Output($\mathcal{S}_{i+1}$) $i \longleftarrow i+1$
    until $\mathcal{S}_{i-1} = \mathcal{S}_i \vee (i > k)$
    return $\mathcal{S}_i$
end
```

We have to ensure that Knuth-Bendix completion yields a useful result. In the best case, it should yield a complete rewriting system and at least it should give unique residues for strings of bounded lengths. We cannot hope to tell if the procedure stops at any point until it did.

For this we characterise the rules that are needed for a set of rewrite rules to form a complete set. These rules are called $\mathcal{S}$-*essential* rules.

Definition 3.10 (essential rules):
*Let $\mathcal{S}$ be a rewriting system. Then a rule $v \to w$, not necessarily contained in $\mathcal{S}$, is $\mathcal{S}$-*essential *if w as well as all proper substrings of v are $\mathcal{S}$-irreducible and $\mathcal{S}$-minimal.* □

This means that during completion of $\mathcal{S}$, the rules that stay in $\mathcal{S}_i$ are the $\mathcal{S}$-essential rules. Running Knuth-Bendix completion for a sufficiently long time guarantees that all essential rules will be generated. Unfortunately sufficiently long might mean infinitely long and we do rarely have that much time on our hands. In some cases, the procedure terminates with a finite complete set of rewrite rules, and sometimes the rewrite rules have a special structure that is reasonably easy to describe. We prove that running the completion process for long enough guarantees that the result is in fact a complete set of rewrite rules.

Theorem 3.11 (completion works):
Let $\mathcal{S}$ be a string rewriting system over A.

1) *For all s in A^* there is an n_0 such that s is reduced to the minimal element v in $\leftrightarrow^*$ using $\mathcal{S}_{n_0}$ from Algorithm 4.*

2) *If $v \to w$ is an $\mathcal{S}$-essential rule, then there is an n_0 such that $\mathcal{S}_{n_0}$ contains $v \to w$ as a rewrite rule.*

3) *For a set $\mathcal{S}_n$ of rewrite rules produced by Algorithm 4 there is a k such that all strings of length at most k are rewritten to their unique minimal representative in their congruence class.*

Proof: Let

$$\widetilde{\mathcal{S}} := \bigcup_{i=0}^{\infty} \bigcap_{j=i}^{\infty} \mathcal{S}_i$$

be the set of all rewrite rules that stay in the rewriting system. That is, all rules that are either there in the first place or added during completion and not removed by the simplification process. This set will most likely be infinite and we can thus not compute it completely.

This means that for each pair of rules in $\widetilde{\mathcal{S}}$ the assumptions of Lemma 3.6 hold. Simplification does not change the relation $\rightarrow^*$, and thus the rewriting system $\widetilde{\mathcal{S}}$ is locally confluent by *Lemma* 3.6. Thus, because $\widetilde{\mathcal{S}}$ is Noetherian, it is complete.

Because $\widetilde{\mathcal{S}}$ is complete, every string s is reduced to the unique residue in its congruence class. Because all rules needed to reduce s are produced at some time during the completion the existence of n_0 in 1) follows.

If $v \rightarrow w$ is an $\mathcal{S}$-essential rule, v is reduced to w by $\widetilde{\mathcal{S}}$. Because all proper substrings of v are minimal, v has to be the left hand side of a rule $v \rightarrow w'$, and since w is minimal $w' \rightarrow^* w$. The next time the set of rules is simplified, the rule $v \rightarrow w$ is added to the set of rewrite rules, and thus the n_0 in 2) exists.

Eventually, because there are only finitely many strings of length k, at some point there are enough rules in $\mathcal{S}_n$ to match all reducible substrings. Thus each string of length k is rewritten to a unique residue.

■

If there are only finitely many equivalence classes under $\leftrightarrow^*$ then the Knuth-Bendix procedure will halt with a finite complete set of rewrite rules. If the monoid presentation is one for an automatic group, the set of essential rewrite rules is regular, and thus we might be able to describe this set using a finite state automaton.

The above treatment should be enough to now continue to the algorithm that actually computes an automatic presentation for a finitely presented group $\mathfrak{G}$. Note that to implement Knuth-Bendix completion efficiently, we will need to employ implementation details as sorting the rewrite rules, such that long substrings are reduced fast to shorter ones and efficient string matching algorithms. This is beyond the scope of this thesis, but [Sim94] goes into some details about this in the context of finitely presented groups.

3.3 The Automatic Groups Algorithm

We now turn to the main task of this chapter. Given a finite group presentation, we want to compute an automatic presentation for the given group. This problem is undecidable in general. But we know that we are dealing with a semi-decidable problem. Thus, if we find a method to produce a set of automata for a group presentation that might constitute an automatic presentation, we can use the axioms given in Section 2.3 to check whether the computed automata correctly are correct. We will come up with a slightly more efficient way of producing automatic presentations than just enumerating automata. The algorithm to be introduced here is due to Derek F. Holt and David B. Epstein and there is an implementation of the algorithm available at [Hol94]. Furthermore, the author plans to implement his ideas of an altenative way to find word acceptors in the near future.

We attempt to compute a **ShortLex** automatic structure for a finitely presented group $\mathfrak{G}$. We will first give an overview of the procedure, then go into a few details concerning the steps. After that we show that the procedure terminates if its input is a group that allows for a **ShortLex** automatic presentation and that the computed presentation is correct.

The practical implementation the author knows of employs a few tricks to make the computation more efficient. The reader is referred to the books [HEO05], [EPC$^+$92] and [Sim94] for a complete account of the algorithms involved. In particular, one does not need to do full axiom checking in every iteration of the algorithm.

In the book [HEO05] Derek Holt gives a comprehensible account of the algorithm itself, in [EPC$^+$92] all the theory about the automatic groups algorithm is taken care of and [Sim94] gives additional relevant methods for finite state automata and rewriting systems. Additionally there is a large body of literature concerning regular languages and finite state automata and concerning term rewriting systems.

The presented automatic groups algorithm has been implemented by Derek Holt and can be used stand-alone from a shell or from the computer algebra system GAP. It can be downloaded from [Hol94].

The behaviour of the procedure has not been thoroughly examined, some of the variables are determined by experiments. In particular there is no good estimate for when to interrupt Knuth-Bendix completion.

Let from now on $\mathfrak{G} := \mathrm{Mon}\langle A \mid R \rangle$ be a finitely presented group presented as a monoid. By convention, A is closed under taking formal inverses and R contains relators (xX, ε) and (Xx, ε) for each x in A. Let $\mathcal{S}_0$ be the string rewriting system that results from R. That is, we make every generating pair (v,w) into a rule $v \rightarrow w$ if $w < v$ in the **ShortLex** order or $w \rightarrow v$ if $v < w$ in the **ShortLex** order.

We assume $\mathfrak{G}$ to be **ShortLex** automatic with respect to the given presentation and assume $\mathfrak{a} = \left(A, \pi, \mathfrak{W}, (\mathfrak{M}_x)_{x \in A \cup \{\varepsilon\}}\right)$ to be an automatic presentation for $\mathfrak{G}$. We run the procedure and compute approximate automatic presentations, hoping that at some point the computed structure is a model of the axioms given in Section 2.3. Automata that turn up during the computation are denoted by $\mathfrak{Z}'$, $\mathfrak{W}'$, $\mathfrak{M}'_x$.

Consider Algorithm 5, which is a slightly simplified version of the automatic groups algorithm by Derek Holt. The subprocedures that are used are not all given in pseudocode, but have names that should be self-explanatory. We will go through the steps and discuss some details. There are certainly many ways to improve the procedure. The first important question that tips up is when to stop the Knuth-Bendix completion process. There is no known criterion for how to choose the parameter k. In general, we have no means of knowing if or when Knuth-Bendix completion will terminate. Even if it would actually terminate some time, we might run out of space or time first. No good theoretical measure has been found to date to estimate when we computed enough word differences to be able to compute an automatic presentation in the following steps. Experiments by researchers showed that it is often practical to interrupt the completion procedure when certain rules only tip up and are removed by the simplification procedure afterwards. The number of word differences that arise from the set of rewrite rules gives another hint. But to be able to check the word differences, we have to compute them first.

The Knuth-Bendix completion process has been introduced in Algorithm 2. We also introduced a method to check whether the set of rules is complete. If the Knuth-Bendix procedure stops with a finite complete set of rewrite rules, we are subsequently able to construct an automatic presentation for $\mathfrak{G}$.

We now look at how to construct a word difference automaton $\mathfrak{Z}'$ from a set of rewrite rules. The notion of word differences associated with pairs of strings has been introduced in Section 2.1. Because $v =_{\mathfrak{G}} w$ for all rules $v \rightarrow w$, it is immedeate to look at the set word differences associated with rules in $\mathcal{S}_i$. We are assuming $\mathfrak{G}$ to be automatic, thus we know that the set D of word differences associated with all pairs of strings v and w in $L(\mathfrak{W})$ with $v \otimes w$ in $L(\mathfrak{M}_x)$ is finite. For an $\mathcal{S}$-essential rule $v \rightarrow w$ the set $D(v,w)$ of word differences associated with the pair (v,w) is a subset of D and thus finite. As a corollary the set $D_{\mathcal{S}}$ of all word differences associated with $\mathcal{S}$-essential rules is also finite. Thus, slightly surprisingly, the set of word differences for an automatic group is already defined by a finite

Algorithm 5: ComputeAutomaticPresentation()

Input: $\mathfrak{G} := \mathrm{Mon}\langle\, A \mid R \,\rangle$, tweaking parameter k.
Output: On successful termination, a **ShortLex** presentation for $\mathfrak{G}$.

```
begin
    $i \longleftarrow 1$
    $\mathcal{S}_0 \longleftarrow$ MakeRewritingSystem($R$)
    repeat
        repeat
            $\mathcal{S}_i \longleftarrow$ KnuthBendix2($\mathcal{S}_{i-1}, k$)
            $D'_{\mathcal{S}_i} \longleftarrow$ ComputeWordDifferences($\mathcal{S}$)
            $i \rightarrow i+1$
        until WordDifferencesStable($D'_{\mathcal{S}_i}$)
        $\mathfrak{Z}' \longleftarrow$ ConstructWordDifferenceSystem($D'_{\mathcal{S}_i}$, $\mathcal{S}_i$)
        $\mathfrak{W}' \longleftarrow$ ConstructWordAcceptor($\mathfrak{Z}'$)
        repeat
            foreach $x \in A \cup \{\varepsilon\}$ do
                $\mathfrak{M}'_x \longleftarrow$ ConstructMultiplier($\mathfrak{Z}'$, $x$)
                if not MultiplierCorrect($\mathfrak{M}'_x$) then
                    AddWordDifferences($D'_{\mathcal{S}_i}$)
                    $\mathfrak{Z}' \longleftarrow$ConstructWordDifferenceSystem($D'_{\mathcal{S}_i}$, $\mathcal{S}_i$)
                end
            end
        until MultipliersCorrect($(\mathfrak{M}_x)_{x \in A \cup \{\varepsilon\}}$)
    until CheckIfCorrect($\mathfrak{G}$, $\mathfrak{W}'$, $\mathfrak{M}'_\varepsilon$, $(\mathfrak{M}'_x)_{x \in A}$)
    return $\left(\mathfrak{W}', \mathfrak{M}'_\varepsilon, (\mathfrak{M}'_x)_{x \in A}\right)$
end
```

subset of the set of $\mathcal{S}$-essential rules. Our problem is that we do not know which subset that is.

If $v \rightarrow w$ is an $\mathcal{S}$-essential rule, then $v =_{\mathfrak{G}} w$ and w and all proper infixes u of v are $\mathcal{S}$-irreducible. Thus assuming $v = ux$ for u in A^* and x in A, the string $u \otimes w$ is accepted by $\mathfrak{M}_x$. All word differences associated with u and w are also associated with v and w thus $D(v,w)$ has to be a subset of D.

From the knowledge that the word differences associated with $\mathcal{S}$-essential rules form a subset of the word differences associated with all pairs of strings $v \otimes w$ such that $vx =_{\mathfrak{G}} w$, we deduce that we should attempt to compute a word difference system $\mathfrak{Z}'$ for $\mathfrak{G}$ using rules from $\mathcal{S}_i$. We first give an algorithm to compute a finite state automaton $\mathfrak{Z}'$ that accepts word differences associated with rules in $\mathcal{S}$. After that we can compute a word acceptor for an automatic presentation. From there on it is only a small step to computing multiplication automata.

Algorithm 6: ComputeWordDifferences()

```
    Input: A finite set $\mathcal{S}$ of rewrite rules.
    Output: The set $D'_{\mathcal{S}}$ of word differences associated with all rules in $\mathcal{S}$.
1   begin
2       $D'_{\mathcal{S}} \longleftarrow \emptyset$
3       foreach $v \rightarrow w$ in $\mathcal{S}$ do
4           $d_0 \longleftarrow \varepsilon$
5           for $1 \leqslant i \leqslant |v \otimes w|$ do
6               $D'_{\mathcal{S}} \longleftarrow D'_{\mathcal{S}} \cup \left\{ \rho_{\mathcal{S}}\left( v[i]^{-1} w[i] \right) \right\}$
7           end
8       end
9   end
```

Algorithm 6 shows how to compute the set of word differences that are associated with rules in a set $\mathcal{S}$ of rewrite rules. It is straightforward. We construct a word difference automaton $\mathfrak{Z}' = \left(Q, A^{\otimes 2}, \{\varepsilon\}, Q, \tau\right)$ in the following way. We take the set of states to be $Q := D_{\mathcal{S}}$, the initial state to be ε and transitions $(v, x \otimes y, w)$ in τ if and only if $w = \rho_{\mathcal{S}}(Xvy)$.

To construct the $\mathfrak{Z}'_x$ for each x in A, we set the set of accept states to $F := \{\rho_{\mathcal{S}}(x)\}$.

We note that the $\mathfrak{Z}'$ might still depend on how we form $\rho_{\mathcal{S}_i}$ if $\mathcal{S}_i$ is not confluent. We take f to be equal to π for all elements of Q and $\mathfrak{Z}'$ is a word difference automaton for $\mathfrak{G}$.

We show that the constructed word difference automaton accepts all $\mathcal{S}$-essential rules if the Knuth-Bendix procedure is run for long enough. The language of $\mathfrak{Z}'$ is then a complete set of rewrite rules.

Lemma 3.12 **($\mathcal{S}$-essential rules accepted):**
For the set $\mathcal{S}$ of rewrite rules, there is a number n_0 in $\mathbb{N}$ such that after n_0 iterations of the Knuth-Bendix completion process, $\mathfrak{Z}'_{\varepsilon}$ will accept all $\mathcal{S}$-essential rules $v \rightarrow w$.

Proof: Let $D_{\mathcal{S}}$ be the set of all word differences that are associated with $\mathcal{S}$-essential rules. We know, that $D_{\mathcal{S}_i}$ is finite and a subset of D, the set of all word differences associated with pairs of strings such that $v \otimes w$ is accepted by one of the $\mathfrak{M}_x$.

Because $D_{\mathcal{S}}$ is finite, there is a finite subset $\mathcal{T}$ of the set of $\mathcal{S}$-essential rules, such that $D(\mathcal{T})$ is equal to $D_{\mathcal{S}}$. Because $\mathcal{T}$ is finite and Knuth-Bendix completion produces every $\mathcal{S}$-essential rule after a finite amount of time, at some point $\mathcal{T}$ will be a subset of $\mathcal{S}$ during the completion.

Also, after a finite amount of time the canonical reduction of Xwy will be equal to the $\mathcal{S}$-minimal representative of Xwy.

At that point $D'_{\mathcal{S}_i}$ will contain all minimal words representing elements of $D_{\mathcal{S}_i}$. Then the word difference automaton $\mathfrak{Z}'$ will have well defined transitions $(w, x\otimes y, \rho_{\mathcal{S}}(Xwy))$ for all $x\otimes y$ in $A^{\otimes 2}$.

Thus by construction, $\mathfrak{Z}'_\varepsilon$ accepts $v\otimes w$, because there is a well defined run of $\mathfrak{Z}'_\varepsilon$ on $v\otimes w$ and $v =_{\mathfrak{G}} w$. ■

To produce a word acceptor, we define the following regular language. One example of an automatic order is the **ShortLex** order. This is the only point at which we need the reduction order to be automatic. If we find a different procedure to produce a word acceptor we would have freed the procedure of the dependency on an automatic reduction order. We will go into further detail on this in Section 3.4.

$$L := \left\{v \mid \exists w\ v\otimes w \in L\left(\mathfrak{Z}'_\varepsilon\right) \wedge w < v\right\}$$

We take the language W' to be $\overline{A^*LA^*}$, the language of strings that do not have any reducible infix. This language is prefix-closed and regular. Using the constructions introduced in Section 1.4 we can construct an automaton $\mathfrak{W}'$ with language $L(\mathfrak{W}') = W'$. We want to prove that if the Knuth-Bendix procedure is run for long enough, we produce a correct word acceptor in this step, that is $L(\mathfrak{W}) = L(\mathfrak{W}')$.

Lemma 3.13 (approximating $\mathfrak{W}$):
The language $L(\mathfrak{W})$ is a subset of $L(\mathfrak{W}')$ with equality if and only if $\mathfrak{Z}'_\varepsilon$ accepts all $\mathcal{S}$-essential rules.

Proof: We show that a string that is not accepted by $\mathfrak{W}'$ is also not accepted by $\mathfrak{W}$. Let s not be accepted by $\mathfrak{W}'$. Then, by definition of $\mathfrak{W}'$, the string s is an element of A^*LA^*, that is $s = s_1vs_2$ and there is a string w such that $v\otimes w$ is accepted by $\mathfrak{Z}_\varepsilon$. By the definition of a word difference automaton $v =_{\mathfrak{G}} w$. Thus $s =_{\mathfrak{G}} s_1ws_2$ and $s \geqslant s_1ws_2$ thus s is not accepted by $\mathfrak{W}$, because it is not a **ShortLex** minimal representative.

To show equality if $\mathfrak{Z}_\varepsilon$ accepts all $\mathcal{S}$-essential rules, assume that s not accepted by $\mathfrak{W}$. Then s cannot be minimal, thus there are substrings of s that are not minimal. Let v be the shortest substring of s that is not minimal. Then there exists a w in A^* such that $v =_{\mathfrak{G}} w$ and w is minimal. Then $v \to w$ is an $\mathcal{S}$-essential rule. Thus, if $\mathfrak{Z}_\varepsilon$ accepts all essential rules, $v\otimes w$ is accepted by $\mathfrak{Z}_\varepsilon$ and s is not accepted by $\mathfrak{W}'$.

■

From this we deduce that if we run Knuth-Bendix completion for long enough, we will get a correct word acceptor for $\mathfrak{G}$. Because by Lemma 3.12 the automaton $\mathfrak{Z}_\varepsilon$ accepts all $\mathcal{S}$ essential rules if the Knuth-Bendix completion is run for sufficiently long.

From here on we can thus assume that we have found a correct word acceptor. We construct multiplication automata $\mathfrak{M}'_x$ for x in A by constructing automata accepting the languages $L(\mathfrak{W}')\otimes L(\mathfrak{W}')\cap L(\mathfrak{Z}'_x)$ for each $x \in A$. These languages are regular by Corollary 1.38.

Assuming we found a correct $\mathfrak{W}'$, we check whether $\mathfrak{M}'_x$ is correct. For this we make ourselves clear what happens if some $\mathfrak{M}'_x$ is not correct. From the construction of $\mathfrak{M}'_x$ it follows that if $v\otimes w$ is accepted by $\mathfrak{M}'_x$ then $vx =_{\mathfrak{G}} w$. Thus, if for some $x \in A$ the automaton $\mathfrak{M}'_x$ is not correct, there has to be a string v that is an element of $\mathfrak{W}'$ such that there is no w in $L(\mathfrak{W}')$ such that $v\otimes w$ is accepted by $\mathfrak{M}'_x$. This is the case because $\mathfrak{W}'$ is constructed to be a unique word acceptor and we assume $\mathfrak{W}'$ to be correct. We express correctness of $\mathfrak{M}'_x$ by an FO$[M_x]$ formula $\varphi(w) = \exists v M_x(v,w)$. Thus there is an automaton that accepts the language consisting of exactly the strings w for which $\varphi(w)$ holds. If this language is equal to A^*, then $\mathfrak{M}'_x$ is correct. If there is a string v that is accepted by the automaton for $\neg\varphi(w)$, we use the computed word difference system $\mathfrak{Z}'_x$ to compute a string w such that $vx =_{\mathfrak{G}} w$. For this pair of strings, we again compute word differences and add them to the set D'_S. We start over with the procedure to compute multiplication automata.

In the last step we use the map η introduced in Section 1.4 to construct an automaton for the conjunction of the axioms given in Section 2.3 using the automata $\mathfrak{W}'$ and $\mathfrak{M}'_x$ for $x \in A \cup \{\varepsilon\}$. We check if the resulting automaton accepts the empty tuple. If it does, the computed presentation is correct.

We still need to prove that if the given group allows for a **ShortLex** automatic presentation and given enough time and memory, the procedure will terminate with an automatic presentation for $\mathfrak{G}$. For this we note that the set of word differences for a group is encoded in a finite set of rules, the problem being that we do not know which subset that is.

Theorem 3.14 (successful termination if ShortLex-automatic):
Let $\mathfrak{G}$ be a finitely presented group. If $\mathfrak{G}$ is **ShortLex** *automatic and the Knuth-Bendix completion process is run for sufficiently long, then Algorithm 5 does compute an automatic presentation for $\mathfrak{G}$.*

Proof: We let the Knuth-Bendix completion procedure run long enough to fulfil the hypothesis of Lemma 3.12 and Lemma 3.13. Thus we assume we computed a correct word acceptor and we assume that the Knuth-Bendix procedure is run long enough to ensure that all strings in D'_S are reduced to their minimal representatives by the rewriting system $\mathcal{S}$ and that all Xwy that are needed to produce a correct word difference system are also reduced to their minimal representatives.

In Theorem 3.11 we showed that every $\mathcal{S}$-essential rule turns up in the course of Knuth-Bendix completion after a finite amount of time. We showed that word differences associated with $\mathcal{S}$-essential rules are elements of D, the set of all word differences associated with pairs of strings v and w such that $vx =_{\mathfrak{G}} w$ for an x in A. Because we assume $\mathfrak{G}$ to be **ShortLex** automatic, D is finite. Thus all word differences associated with $\mathcal{S}$-essential rules will turn up after a finite amount of time.

If D'_S contains minimal representatives for all elements of D then the word difference automaton $\mathfrak{Z}'$ will be correct and the $\mathfrak{M}'_x$ for $x \in A \cup \{\varepsilon\}$ will also be correct. The algorithm then proceeds to the final model-checking step.

If we do not have all representatives for elements of D in D'_{S_i}, we will add them in the step that checks the multipliers for correctness. Because D is finite, this process will terminate after finitely many iterations. We will then have computed a correct automatic presentation for $\mathfrak{G}$.

■

3.4 An Alternative Algorithm to Compute Word Acceptors

This section is dedicated to the development of an alternative algorithm to compute word acceptors for automatic groups. It appeared that for theoretical purposes the word acceptors produced by the **kbmag** package seemed not very friendly. Thus there had to be an alternative way to describe them. In an attempt to make the word acceptors more tractible, the states of the word acceptors for triangle groups produced by the **kbmag** procedure were labelled by the shortest path in the automaton that lead to them. These labels are shortest representatives of the Nerode congruence classes of the language of the word acceptor.

It seems much more convenient for some theoretical purposes to describe the Nerode congruence classes of word acceptors directly by using a complete rewriting system. The proposed algorithm has two main advantages over the **kbmag** procedure. First, we can compute the minimal acceptor directly from the set of rewrite rules and second, we can use reduction orders that are not automatic. However, it is not yet clear whether this leads to a true improvement of the procedure. The existing procedure needs at least one powerset construction and can only use automatic orders. The **ShortLex** order always seems a bit unnatural for groups. However, the proposed procedure still has to be implemented and might turn out to be impractical, if only for the large amounts of string processing needed.

This algorithm might also be able to compute word acceptors for groups that are not automatic but regularly generated, enableing us to at least have a method to enumerate group elements. It also seems that this procedure is related to the computation of cone types as defined by Cannon and dealt with in [EPC$^+$92] Chapter 3. Thus, when using this procedure for general groups it might be that the produced word acceptor does not belong to an automatic presentation of the group even if the group is automatic. For word hyperbolic groups we are safe, because for word hyperbolic groups the produced word acceptor is one that is contained in an automatic presentation. This is also shown in Chapter 3 of [EPC$^+$92].

As a preparation we repeat the description of the language of the word acceptor given in Section 3.3. First we defined the language L as follows.

$$L := \left\{ v \mid \exists w\, v \otimes w \in L\left(\mathfrak{Z}'_\varepsilon\right) \wedge v \leqslant w \right\}.$$

Where $\mathfrak{Z}'$ is the computed word difference automaton. This is the language of left hand sides of $\mathcal{S}$-essential rules.

Then we gave the language W by

$$W := \overline{A^* L A^*}.$$

Thus W is the language of strings over A that do not have an infix that is reducible under the rewriting system.

We also repeat the definition of the Nerode congruence as this is important for the following description. Given a language L, we define two strings v and w to be congruent if and only if

$$vu \in L \Leftrightarrow wu \in L \text{ for all } u \in A^*.$$

We denote the Nerode congruence classes of a language by $[v]_L$ and also write $v \equiv_L w$ for $[v]_L = [w]_L$. We know L is regular if and only if there are only finitely many Nerode congruence classes and that these classes describe the deterministic finite state automaton with language L and minimal state count.

We now want to look at how the Nerode congruence can be relevant in the computation of a word acceptor for a group. By the definition above, letting for the moment aside that the languages L and W should be regular, we want to tell when a string over A is a $\leqslant$-minimal representative of a Nerode congruence class. We also want to be able to tell, if two strings v and w belong to the same W congruence class or not.

We can equivalently describe either the Nerode congruence classes of the language of left hand sides of $\mathcal{S}$ essential rules or the Nerode classes of W. We decide to describe W directly here.

We need tools to tell whether two strings belong to the same Nerode congruence class. If the string vx is a representative of a Nerode congruence class, then vxX is not accepted by the word acceptor, because this string is reducible. If the sink state of an automaton exists then it is unique. Thus for every representative of a congruence class that ends in some generator x the transition with label X goes to the sink state.

Lemma 3.15 (Nerode congruence classes for automatic presentations):
Let $\mathcal{C}$ be a finite reduced rewriting system over an alphabet A and let $\overline{\mathcal{C}}$ be its completion. If the language W of strings irreducible under $\overline{\mathcal{C}}$ is regular, then we can compute all of the Nerode congruence classes

of W if there is an $n \in \mathbb{N}$ dependent on W such that all strings irreducible under $\overline{\mathcal{C}}$ of length n are already reduced to their unique irreducible residue by $\mathcal{C}$.

Proof: We know that if we let the Knuth-Bendix completion process run for long enough, we will eventually get a set of rewrite rules $\mathcal{C}$ that reduces strings of length n to their irreducible residue under $\overline{\mathcal{S}}$. Because we assume W to be regular, there are only finitely many different Nerode congruence classes and these classes each have a representaive of minimal length under the reduction order. Each pair of classes represented by strings v and w is seperated by a string s of finite length. That is, vs is not an element of W and ws is or vice versa. This means that vs is reducible. Because $\mathcal{C}$ is assumed to be finite we can try all possibilities to extend v to a left hand side of a rule in s. ■

There are some considerations we can make to make the procedure practical. If we are given two strings v and w over A and want to test whether $[v]_W = [w]_W$ we search for rules in $\mathcal{S}$ such that a postfix of v is a prefix of a left hand side of the rule. We can then extend v by a string v' such that vv' contains the left hand side of this rule as postfix. If wv' is irreducible the congruence classes are not the same. Unfortunately we have to perform this for every possible extension of v and w to left hand sides of rules in $\mathcal{S}$.

We describe the procedure to produce a word acceptor from a given set $\mathcal{S}$ of rewrite rules and a reduction order $\leqslant$. For this we create a table that has $|A|+1$ columns, where A is the set of monoid generators for the group $\mathfrak{G}$. The first column contains representatives of Nerode congruence classes. We start with two rows: one for the empty congruence class $[]$ and one for the congruence class of the empty word $[\varepsilon]$. We assume the language not to be empty, because in that case we would not even get a surjective map onto the group.

We mark the congruence class of the empty word as *open*. In every step of the algorithm we pick a representative r of a congruence class that is marked as open. We decide to pick the least one in the reduction order. We in turn append every generator x in A to r, which yields the string rx. We have to prove for rx that it is either contained in one of the previously produced congruence classes or that it is different from all the previously produced ones. If the representative computed is one for a new class, we add a new row to the table and mark the representative as open. We also put the entry into the table in the column for the current generator. If the representative belongs to a class found before, we put the representative of this class into the table. If we are done with r, we mark it as *closed*. The algorithm terminates if there are no open congruence classes left. This algorithm always terminates if we assume the set $\mathcal{S}$ to be finite. If $\mathcal{S}$ is not complete the computed result will in some cases be incorrect even if the algorithm terminated. For the time being we rely on the axioms to check for correctness.

The result of the algorithm is a transition table. The initial state is the state $[\varepsilon]$ and the set of final states consists of all states except for the sink state $[]$.

In [EPC$^+$92] the *cone type* $C(s)$ of a string s is defined as the set of strings t such that st is a geodesic. When talking about languages of geodesics this coincides with the definition of the Nerode congruence, because the cone type of a string s contains exactly the strings that are Nerode congruent to s. The situation needs a bit more investigation but this is not in the scope of this thesis. We also note the similarity between the above algorithm and the Todd-Coxeter enumeration procedure.

Algorithm 7: ComputeWordAcceptor()

Input: Set $\mathcal{S}$ of rewrite rules.

Output: Finite state automaton $\mathfrak{W}'$ with language $W' = \overline{A^*LA^*}$.

```
begin
    foreach $x \in A$ do
        table[][$x$] $\longleftarrow$ []
    end
    table[$\varepsilon$] $\longleftarrow$ open
    while table contains open rows do
        $r \longleftarrow$ representative in open row
        foreach $x \in A$ do
            if $rx$ represents a new congruence class then
                table[$rx$] $\longleftarrow$ open
            else
                table[$r$][$x$] $\longleftarrow$ [$rx$]
            end
        end
        table[$r$] $\longleftarrow$ closed
    end
end
```

4 Triangle Groups

"It is indeed wonderful that so simple a figure as the triangle is so inexhaustible in properties. How many as yet unknown properties of other figures may there not be?"

(August Crelle)

In this chapter we introduce the triangle groups $\Delta(p,q,r)$. Triangle groups are a class of geometrically motivated groups and we have a good understanding of the Cayley graph of a triangle group. The triangle groups discussed here are groups of symmetries of geometric spaces that fix regular tilings of that space by triangles. German readers might like to read [Ros04] to get a very accessible account of the topic. Johnson in [Joh80] also treats triangle groups and their finite presentations. Most results in this chapter might well extend to Coxeter groups which are generalised triangle groups.

The motivation to look at small and easy examples for automatic groups is that it is to date not entirely clear how the automatic group algorithm reacts to differing inputs. It is not known how or if confluence of a set of rewrite rules interplays with automaticity and also changes to the generating set of a group were not considered to date.

We start by defining triangle groups by a finite presentation and we will for the most part be ignoring deeper geometric properties of these groups.

Definition 4.1 (triangle group):
Let (p,q,r) be a triple of natural numbers each greater than 1. Then the (p,q,r)-triangle group $\Delta(p,q,r)$ is defined by the following finite group presentation

$$\Delta(p,q,r) := \langle\, x,y \mid x^p, y^q, (xy)^r \,\rangle,$$

or the equivalent monoid presentation

$$\Delta(p,q,r) := \mathrm{Mon}\langle\, x,y,X,Y \mid (xX,\varepsilon),(Xx,\varepsilon),(yY,\varepsilon),(Yy,\varepsilon),(x^p,\varepsilon),(y^q,\varepsilon),((xy)^r,\varepsilon) \,\rangle. \qquad \square$$

We choose p, q and r to be greater than 1, because if one of the generators has order 1 we would end up with a cyclic group. Triangle groups are in general not isomorphic to a cyclic group. This is easily seen because in all non trivial cases these groups are not abelian, because $xy = (YX)^{r-1}$ which is not equal to yx.

Another question that arises when looking at the given presentation is whether the isomorphism type of the presented group depends on the order of the parameters. Quite obviously $\Delta(p,q,r)$ and $\Delta(q,p,r)$ are isomorphic. The following lemma shows that indeed the isomorphism type of the presented group does not depend on the order of p, q and r.

Lemma 4.2 (invariance):
The isomorphism type of a triangle group $\Delta(p,q,r)$ is invariant under permutation of the paramters.

Proof: It is quite obvious that we can exchange p and q in the above presentation. This leaves us with the case that we can exchange p with r. We give an isomorphism from $\Delta(p,q,r)$ to $\Delta(p,r,q)$. Let

$$\Delta(p,q,r) = \langle\, x,y \mid x^p, y^q, (xy)^r \,\rangle$$

and

$$\Delta(p,r,q) = \langle\, a,b \mid a^p, b^r, (ab)^q \,\rangle$$

We define φ to map x to a^{-1} and y to ab and extend φ to a homomorphism between finitely presented groups. This homomorphism is clearly bijective and thus an isomorphism. ■

Thus, we can as a convention, let $p \geqslant q \geqslant r$ for the time being. Note at this point that the a computed automatic presentation might very well depend on the order of p, q and r as well as the order of the set of generators that we choose.

Triangle groups can be finite or infinite. The space in which the tiling lives is thus dependent on the parameters p, q and r. We define

$$\rho(p,q,r) := \frac{1}{p} + \frac{1}{q} + \frac{1}{r}.$$

Depending on the values of p, q and r there are three cases, namely the *elliptic* case for $\rho(p,q,r) > 1$, the *euclidean* case for $\rho(p,q,r) = 1$ and the *hyperbolic* case for $\rho(p,q,r) < 1$. These cases correspond to the triangles having an angle sum of less than, equal to or greater than π. Elliptic triangle groups are finite and in the remaining two cases the groups are infinite.

We move on to give the sets of parameters for which the three cases arise.
We assume $r = 2$, because if r was greater than 2 this would leave us with $\rho(p,q,r) \leqslant 1$. For $q = 2$ we can arbitrarily choose p. For $q = 3$ there are three remaining cases with $p \in \{3,4,5\}$ and these are also all cases that can arise. We give a table with the isomorphism types of elliptic triangle groups in Table 4.1. These groups are trivially automatic as they are finite.

(p,q,r)	iso. type	order
$(p,2,2)$	D_p	$2p$
$(3,3,2)$	A_4	12
$(4,3,2)$	S_4	24
$(5,3,2)$	A_5	60

Table 4.1: Isomorphism types of elliptic triangle groups

For the euclidean case we get just three solutions to the equation $\rho(p,q,r) = 1$. These are $(6,3,2)$, $(4,4,2)$ and $(3,3,3)$ and these three groups are infinite. We will not prove this result here but we will see later on that these groups are in fact infinite. The euclidean case might be the most intuitive one as the tiling can be visualised in the euclidean plane. For example the $(3,3,3)$-triangle group is the group of translations and rotations that leaves a tiling of the euclidean plane by equilateral triangles invariant.

The hyperbolic case takes all other possible parameters. These groups are also infinite and act on the hyperbolic space. Quite a few models have been proposed for the hyperbolic plane. One is due to

Poincaré. The Poincaré disc model visualises the hyperbolic plane by a unit disc in $\mathbb{R}^2$. We will not go into further geometric detail in this thesis. The results of the computational experiments show that there seem to be no connections between the groups being euclidean or hyperbolic and properties of the rewriting system.

We are mainly interested in the cases where the groups considered are infinite. If we have an automatic presentation for an infinite group, we have tools to work computationally with these groups.

4.1 Computational Experiments

We now want to look at automatic presentations for triangle groups and how they are generated by the automatic groups algorithm. It is already known that (p,q,r)-triangle groups are automatic, but we will also be able to derive this from the results we show in a later section.

The author conducted a few computational experiments using the **kbmag** package. For this a set of input files for the **kbmag** package was produced and afterwards the autgroup script was called for each of the input files, producing a complete automatic presentation for the given group. After that it turned out that it would be useful to also have the rewriting systems the kbprog program produced in the course of computing the automatic presentation, thus these were also generated.

Figure 4.1 shows an example of an input file for the $(6,4,2)$-triangle group and the word acceptor as it is produced by the automatic groups procedure.

The set of parameters considered includes the tuples (p,q,r), the generating set of the group and the ordering of these generators. Thus the parameter space is quite large and small steps had to be taken to get meaningful results.

In the first step we fix the monoid generating set to be

$$A := \{x, y, X, Y\}$$

and the order on that generating set to be $x < y < X < Y$.

An associated monoid presentation is then given by the generating set A and the following set of relations.

$$R := \{(xX,\varepsilon),\ (Xx,\varepsilon),\ (yY,\varepsilon),\ (Yy,\varepsilon),\ (x^p,\varepsilon),\ (y^q,\varepsilon),\ ((xy)^r,\varepsilon)\}.$$

The first set of parameters included p, q and r with $100 \geqslant p \geqslant q \geqslant r \geqslant 2$ keeping the ordering $x < y < X < Y$ on A fixed. It is quite clear from the results that the number of states and the number of transitions in the word acceptors increases linearly in p, q and r.

The following experiments did not involve computing the automatic presentations for all $100 \geqslant p \geqslant q \geqslant r \geqslant 2$, because this would have taken far too long and would not have yielded additional information. It turned out that residual cases occur only when the parameters were very small, that is $p < 6$.

Changing the ordering on A to $x < X < y < Y$ yields different state counts in the word acceptors, but the variation is not very large. Also, in contrast to the case where $x < y < X < Y$, a complete set of rewrite rules is infinite. Looking at the set of word differences it turns out that changing the ordering does not seem to change the set of word differences. This also explains why the state count in the word acceptors does not change much. Changing the ordering of A^* to the wreath-product ordering resulted in the occurrence of a bug in **kbmag**: the program crashed with a notice to contact the author. This case was thus not investigated further.

The next parameter changed was the presentation, adding one generator z,

$$A' := \{x,y,z,X,Y,Z\}$$

with order $x < y < z < X < Y < Z$ and changing the relations to

$$R' := \{(xX,\varepsilon),\ (Xx,\varepsilon),\ (yY,\varepsilon),\ (Yy,\varepsilon),\ (zZ,\varepsilon),\ (Zz,\varepsilon),\ (x^p,\varepsilon),\ (y^q,\varepsilon),\ (z^r,\varepsilon),\ (xyz,\varepsilon)\}.$$

At a first glance there seems to be no additional value except for the word acceptors being slightly smaller. When we have seen how the word acceptors can be described it turns out that the word acceptors are very similar.

Resulting from these experiments is a huge amount of automatic presentations that were computed on a Pentium class PC in the course of a few weeks using simple bash and Haskell scripts. However, at first glance there is not too much information that can be read from them. To explore the structure of the word acceptors more thoroughly, we turn to the steps involved in the algorithm introduced in Section 3.3 and look at what happens there, hoping to sched more light on the results.

4.2 Monoid Rewriting Systems for the Triangle Groups

Because the first step of the algorithm introduced in Section 3.3 involves Knuth-Bendix completion, we first take a look at the rewriting systems involved in the construction of automatic presentations for triangle groups. Fortunately in the case of the triangle groups we are able to find presentations that allow for a finite complete rewriting system. Thus we are easily able to construct the word difference acceptor $\mathfrak{Z}$ and a complete automatic presentation for the groups.

We can thus conclude that triangle groups are automatic, because we find an automatic presentation for them for arbitrary p, q and r greater than 1 in $\mathbb{N}$.

Showing that we can in fact give a finite complete rewriting system for all triangle groups is not as hard as it seems, but it is a rather tedious task to verify that the given set of rewrite rules is complete. Remember that we have to consider every possible overlap between left-hand sides of rules and check that the local confluence condition holds.

We start by showing that a finitely presented cyclic group has a complete rewriting system. Cyclic groups also have the nice property that we only have one generator to take care of and thus do not need to take care of alphabet ordering issues.

Lemma 4.3 (cyclic groups have finite rws):
Let p be a natural number. We present a finite cyclic group of order p by

$$\mathfrak{C}_p := \mathrm{Mon}\langle\, x,X \mid xX = \varepsilon, Xx = \varepsilon, x^p = \varepsilon \,\rangle.$$

The associated monoid rewriting system $\mathcal{C}$ contains the following rules

$$\begin{array}{lll} (i_{x,1}) & xX & \to & \varepsilon \\ (i_{x,2}) & Xx & \to & \varepsilon \\ (p_x) & x^p & \to & \varepsilon. \end{array}$$

There is a finite complete set $\mathcal{C}_p$ for $p \in \mathbb{N}$, containing the rules

$$\begin{array}{llll} (i_{x,1}) & xX & \to & \varepsilon \\ (i_{x,2}) & Xx & \to & \varepsilon \\ (p_{x,1}) & x^{\lceil \frac{p+1}{2} \rceil} & \to & X^{\lfloor \frac{p-1}{2} \rfloor} \\ (p_{x,2}) & X^{\lceil \frac{p}{2} \rceil} & \to & x^{\lfloor \frac{p}{2} \rfloor} \end{array}$$

```
_RWS := rec(
        isRWS := true,
        generatorOrder := [x,y,X,Y],
        inverses := [X,Y,x,y],
        ordering := "shortlex",
        equations := [
                [x^6, IdWord ],
                [y^4, IdWord ],
                [(x*y)^2, IdWord]
                ]
);

_RWS.wa := rec(
           isFSA := true,
        alphabet := rec(
                type := "identifiers",
                size := 4,
              format := "dense",
               names := [x,y,X,Y]
               ),
          states := rec(
                type := "simple",
                size := 13
               ),
           flags := ["DFA","minimized","BFS","accessible","trim"],
         initial := [1],
       accepting := [1..13],
           table := rec(
              format := "dense deterministic",
      numTransitions := 28,
         transitions := [[2,3,4,5],
                         [6,7,0,5],
                         [8,9,4,0],
                         [0,9,10,0],
                         [11,0,0,0],
                         [12,7,0,5],
                         [0,9,4,0],
                         [6,0,0,5],
                         [8,0,4,0],
                         [0,13,0,0],
                         [12,0,0,5],
                         [0,7,0,5],
                         [0,0,4,0]
                        ]
               )
);
```

Figure 4.1: Input and output of the automatic groups algorithm in the **kbmag** package

.

If we do not want the rounding brackets we can divide this into two cases. For p even, the set $\mathcal{C}_p$ consists of the rules

$$\begin{array}{llcl} (i_{x,1}) & xX & \longrightarrow & \varepsilon \\ (i_{x,2}) & Xx & \longrightarrow & \varepsilon \\ (p_{x,1}) & x^{\frac{p}{2}+1} & \longrightarrow & X^{\frac{p}{2}-1} \\ (p_{x,2}) & X^{\frac{p}{2}} & \longrightarrow & x^{\frac{p}{2}} \end{array},$$

and for p odd $\mathcal{C}_p$ consists of the rules

$$\begin{array}{llcl} (i_{x,1}) & xX & \longrightarrow & \varepsilon \\ (i_{x,2}) & Xx & \longrightarrow & \varepsilon \\ (p_{x,1}) & x^{\frac{p+1}{2}} & \longrightarrow & X^{\frac{p-1}{2}} \\ (p_{x,2}) & X^{\frac{p+1}{2}} & \longrightarrow & x^{\frac{p-1}{2}} \end{array}.$$

Proof: It is fairly easy to apply Algorithm 2 to the set of rewrite rules. What happens here is that the algorithm balances the two sides of the rewrite rule $x^p \longrightarrow \varepsilon$ step by step. In fact this balancing process always takes place, at least for groups and it is in general a good idea to balance left hand sides and right hand sides of the rules before completion. More details on this can be found in [Sim94], Chapter 2, Section 2.7.

We briefly check for confluence by looking at possible overlaps of rules.

The first case is the overlap of rule $(i_{x,1})$ and rule $(p_{x,2})$. We can as well overlap rule $(i_{x,2})$ and rule $(p_{x,1})$ but this case is entirely symmetric to the presented one.

$$\begin{tikzcd} & X^{\lceil\frac{p}{2}\rceil-1} \arrow[r, equal] & X^{\lfloor\frac{p-1}{2}\rfloor} & \\ \underline{\underline{xX}X^{\lceil\frac{p}{2}\rceil-1}} \arrow[ur, "(i_{x,1})"] \arrow[dr, "(p_{x,2})"'] & & & \\ & xx^{\lfloor\frac{p}{2}\rfloor} \arrow[r, equal] & x^{\lceil\frac{p+1}{2}\rceil} \arrow[r, "(p_{x,1})"'] & X^{\lfloor\frac{p-1}{2}\rfloor} \end{tikzcd}$$

The second case deals with the overlap of rule $(i_{x,2})$ and rule $(p_{x,1})$, again the case of rule $(i_{x,1})$ and rule $(p_{x,1})$ is entirely analogous.

$$\begin{tikzcd} & x^{\lceil\frac{p+1}{2}\rceil-1} \arrow[r, equal] & x^{\lfloor\frac{p}{2}\rfloor} & \\ \underline{\underline{Xx}x^{\lceil\frac{p+1}{2}\rceil-1}} \arrow[ur, "(i_{x,2})"] \arrow[dr, "(p_{x,1})"'] & & & \\ & XX^{\lfloor\frac{p-1}{2}\rfloor} \arrow[r, equal] & X^{\lceil\frac{p}{2}\rceil} \arrow[r, "(p_{x,2})"'] & x^{\lfloor\frac{p}{2}\rfloor} \end{tikzcd}$$

We are done, as there are no further overlaps between rules. ■

In the next lemma we show that the free product of finitely many groups that allow for a finite complete monoid rewriting system also has a finite monoid rewriting system. As an immediate consequence, the free product of finitely many cyclic groups allows for a finite complete monoid rewriting system.

Lemma 4.4 (finite free product):
Let $\mathfrak{G}$ and $\mathfrak{H}$ be two finitely presented groups that allow for a finite complete monoid rewriting system. Then the free product $\mathfrak{G} * \mathfrak{H}$ also allows for a finite complete monoid rewriting system.

Proof: Let $\mathfrak{G} = \mathrm{Mon}\langle A \mid R \rangle$ and $\mathfrak{H} = \mathrm{Mon}\langle B \mid S \rangle$ and assume without loss of generality A and B to be disjoint. Then R and S are also disjoint. A presentation for $\mathfrak{G} * \mathfrak{H}$ can be given by $\mathrm{Mon}\langle A \cup B \mid R \cup S \rangle$. We assume $\mathcal{R}$ and $\mathcal{S}$ to be the finite complete sets of rewrite rules for the presentations of $\mathfrak{G}$ and $\mathfrak{H}$. If we form the associated rewriting system for $R \cup S$, we just get the union of the rewriting systems $\mathcal{R}$ and $\mathcal{S}$, and this set is finite and complete. This is because R and S are disjoint and thus there are no overlaps between rules of $\mathcal{R}$ and $\mathcal{S}$ and thus the completeness condition is fulfilled. ■

As a corollary of this lemma, finite free products of groups that allow for a finite complete rewriting system are also automatic.

A (p,q,r)-triangle group is a factor group of the free product of a cyclic group of order p and a cyclic group of order q. Unfortunately, rewriting systems for triangle groups become a bit more complicated. It can even become infinite depending on the ordering of the generators. We will try to develop a finite complete rewriting system for any tuple (p,q,r) of parameters.

We add the rewrite rule $(xy)^r \to \varepsilon$ to the finite complete rewriting system of $\mathfrak{C}_p * \mathfrak{C}_q$ and try to complete the resulting set of rewrite rules. The rules are no longer independent because we created a connection between the two rewriting systems by adding a rule that involves both x and y. To better understand the situation, we first take a look at what happens to the rule $(xy)^r \to \varepsilon$ during completion. It turns out that the situation is not too different from the situation we found in Lemma 4.3.

Lemma 4.5 :
The set

$$\mathcal{C} := \{xX \to \varepsilon,\ Xx \to \varepsilon, yY \to \varepsilon, Yy \to \varepsilon, (xy)^r \to \varepsilon\}$$

of rewrite rules can be completed to the set $\overline{\mathcal{C}}$ containing the following rewrite rules

$$\begin{array}{lll}
(i_{x,1}) & xX & \to \varepsilon \\
(i_{x,2}) & Xx & \to \varepsilon \\
(i_{y,1}) & yY & \to \varepsilon \\
(i_{y,2}) & Yy & \to \varepsilon \\
(c_1) & (xy)^{\frac{r}{2}}x & \to (YX)^{\frac{r-2}{2}}Y \\
(c_2) & (YX)^{\frac{r}{2}} & \to (xy)^{\frac{r}{2}} \\
(c_3) & (yx)^{\frac{r}{2}}y & \to (XY)^{\frac{r-2}{2}}X \\
(c_4) & (XY)^{\frac{r}{2}} & \to (yx)^{\frac{r}{2}}
\end{array}$$

for r even, and

$$\begin{array}{lll}
(i_{x,1}) & xX & \to \varepsilon \\
(i_{x,2}) & Xx & \to \varepsilon \\
(i_{y,1}) & yY & \to \varepsilon \\
(i_{y,2}) & Yy & \to \varepsilon \\
(c_1) & (xy)^{\frac{r+1}{2}} & \to (YX)^{\frac{r-1}{2}} \\
(c_2) & (YX)^{\frac{r-1}{2}}X & \to (xy)^{\frac{r-1}{2}}x \\
(c_3) & (yx)^{\frac{r+1}{2}} & \to (XY)^{\frac{r-1}{2}} \\
(c_4) & (XY)^{\frac{r-1}{2}}Y & \to (yx)^{\frac{r-1}{2}}y
\end{array}$$

for r odd. Note that this case distinction is different from the previous case for one generator.

Proof: The proof involves the same procedure as previous proofs of completeness and is thus omitted.

The sets of rewrite rules bear a certain symmetry, which is due to the convention that the generating set is chosen to be closed under inversion. It would be interesting to show how this symmetry might be relevant in the construction of automatic presentations or Knuth-Bendix reduction. Some further steps of the derivation of the rewrite rules for the triangle groups can be found in Appendix B and finite complete rewriting systems for triangle groups can be found in Appendix C The set of rewrite rules essentially only depends on p, q and r being divisible by two.

4.3 Word Acceptors

In this next step we explore the structure of word acceptors for the triangle groups. It turns out that following the algorithm in Section 3.3 would be tedious and would not yield a satisfactory description of the word acceptors. For example it is not clearly visible how these acceptors relate to the Cayley graph of the group. It is much more convenient and practical to describe the Nerode congruence classes of word acceptors directly by using a complete rewriting system. This approach is described in Section 3.4.

We describe word acceptors for the triangle groups. Note that this description depends on the generating set A we chose. We describe the Nerode congruence classes that we can derive from the rewriting systems we found in Section 4.2 and the transitions and give transition tables. The transitions are almost always obvious. Every node has to have four outgoing transitions, because the transition map has to be total. The set of accept states consists of all states except for the failure state [].

In Table 4.2 we show the Nerode congruence classes and transitions for a word acceptor for the $(6,4,2)$-triangle group that is produced when we run the **kbmag** program for the monoid presentation of this group. The class [] denotes the sink class, once we reach that state, we do not accept any string anymore. The initial state is the state $[\varepsilon]$ and all states except for the sink state are also final states.

From this information we can also read off generators for the syntactic monoid of the word language. The columns for the generators contain the necessary information. The syntactic monoid is an algebraic structure that enables us to look at regular languages more closely. This is beyond the scope of this thesis but it would be interesting to know whether there is any connection between a syntactic monoid of a word acceptor for a triangle group and the group itself.

For the time being we assume $p \geqslant q \geqslant r \geqslant 6$.

As an explanatory example, we also assume p, q and r to be even. Then the set of shortest representatives of Nerode congruence classes is the set of all prefixes of the strings $x^{\frac{p}{2}}$, $X^{\frac{p}{2}-1}$, $y^{\frac{q}{2}}$, $Y^{\frac{q}{2}-1}$, $(xy)^{\frac{r}{2}}$, $(yx)^{\frac{r}{2}}$, $(XY)^{\frac{r}{2}-1}X$ and $(YX)^{\frac{r}{2}-1}Y$. Using this result we can readily give the cardinality of the state sets of the automata.

Most of the transitions are obvious. That is if v and w are representatives of two Nerode congruence classes and $[vx] = [w]$, then there is a transition labelled x from $[v]$ to $[w]$ in the automaton. If right multiplication by a generator yields a reducible string, then $[vx]$ is the sink-class.

This leaves the problem of giving the cases where vx belongs to a congruence class with a shorter representative and deciding which congruence class the string vx belongs to. We have a very small finite set of rewrite rules and can this quite easily determine when we only can extend two strings to the same left hand sides of rules.

Appendix C Section C.1 gives the transition table of a word acceptor for a triangle group when p, q and r are even. We briefly give a few details about some transitions.

		x	y	X	Y
0	$[]$	$[]$	$[]$	$[]$	$[]$
1	$[\varepsilon]$	$[x]$	$[y]$	$[X]$	$[Y]$
2	$[x]$	$[xx]$	$[xy]$	$[]$	$[xY] = [Y]$
3	$[y]$	$[yx]$	$[yy]$	$[yX] = [X]$	$[]$
4	$[X]$	$[]$	$[Xy] = [yy]$	$[XX]$	$[]$
5	$[Y]$	$[Yx]$	$[]$	$[]$	$[]$
6	$[xx]$	$[xxx]$	$[xxy] = [xy]$	$[]$	$[xxY] = [Y]$
7	$[xy]$	$[]$	$[xyy] = [yy]$	$[xyX] = [X]$	$[]$
8	$[yx]$	$[yxx] = [xx]$	$[]$	$[]$	$[yxY] = [Y]$
9	$[yy]$	$[yyx] = [yx]$	$[]$	$[yyX] = [X]$	$[]$
10	$[XX]$	$[]$	$[XXy]$	$[]$	$[]$
11	$[Yx]$	$[Yxx] = [xxx]$	$[]$	$[]$	$[YxY] = [Y]$
12	$[xxx]$	$[]$	$[xxxy] = [xy]$	$[]$	$[xxxY] = [Y]$
13	$[XXy]$	$[]$	$[]$	$[XXyX] = [X]$	$[]$

Table 4.2: Computation of the Nerode classes for the $(6,4,2)$-triangle group's **ShortLex** word acceptor

As stated earlier, adding the formal inverse of the last character in a representative to it results in a reducible string and thus this transition goes to the sink state. This means that in each row of the transition table there has to be at least one transition to the sink state, because we assume the set of generators to be closed under inverses. The largest part of the word acceptor for a triangle group is counting whether a string contains some power of x, y or xy or the respective inverses that is reducible.

There are four interesting transitions that are not quite as we expect them. These are the transitions from $X^{\frac{p}{2}-1}$ to xy, from $Y^{\frac{q}{2}-1}$ to yx and $(XY)^{\frac{r}{2}-1}X$ to yy and $(YX)^{\frac{r}{2}-1}Y$ to xx. As the first two and the second two cases are symmetric we take $X^{\frac{p}{2}}-1$ and $(XY)^{\frac{r}{2}-1}X$ to explain the situation.

Section C.1 shows a finite complete set of rewrite rules for a triangle group with p, q and r even. We look at the possibilites to extend $X^{\frac{p}{2}-1}y$ to a left hand side of a rule. This yields the rules 3, 7, 10 and 14 as candidates. We check whether each extension of $X^{\frac{p}{2}-1}y$ to a string that contains a left hand side of a rule also makes xy reducible. For rule 3 this is easy and rule 7 is as easy as well as rule 10. In each case all that is missing is the leading y. For rule 14 the overlap is rather larger: we take $(xy)^{\frac{r}{2}-1}x$. But this also makes xy reducible. Thus $[X^{\frac{p}{2}-1}y] = [xy]$. Why is $X^{\frac{p}{2}-1}y$ not equivalent to y as one would expect at first? We can find a string that seperates these two classes. We take $(xy)^{\frac{r}{2}-1}x$ as extension. Then $X^{\frac{p}{2}-1}y(xy)^{\frac{r}{2}-1}x$ is reducible, because it is the left hand side of rule 14. But $y(xy)^{\frac{r}{2}-1}x$ is irreducible. It is the right hand side of rule 11.

We look at the Nerode class represented by $(XY)^{\frac{r}{2}-1}X$. Here, rule 15 is important. We want to show that $[(XY)^{\frac{r}{2}-1}Xy] = [yy]$. The rules 3, 7 and 10 are again taken care of easily. For rule 15 we take $y^{\frac{q}{2}-1}$ and end up with reducible strings again.

We now have enough information to compute a word acceptor for the triangle group only given the parameters p, q and r. In particular we can directly give the state count of the automata.

Lemma 4.6 (state count):
For a (p,q,r)-triangle group where $p \geqslant 6$ with monoid presentation

$$\mathrm{Mon}\langle\, x,y,X,Y \mid (xX,\varepsilon),(Xx,\varepsilon),(yY,\varepsilon),(Yy,\varepsilon),(x^p,\varepsilon),(y^q,\varepsilon),((xy)^r,\varepsilon)\,\rangle,$$

the state count size (p,q,r) of the word acceptor is as follows.

$$size\,(p,q,r) = \begin{cases} p+q+4r-6 & p \equiv_2 0, q \equiv_2 0, r \equiv_2 0 \\ p+q+4r-6 & p \equiv_2 0, q \equiv_2 0, r \equiv_2 1 \\ 2p+q+6r-10 & p \equiv_2 1, q \equiv_2 0, r \equiv_2 0 \\ p+2q+6r-10 & p \equiv_2 0, q \equiv_2 1, r \equiv_2 0 \\ 2p+2q+8r-14 & p \equiv_2 1, q \equiv_2 0, r \equiv_2 1 \\ 2p+2q+8r-14 & p \equiv_2 0, q \equiv_2 1, r \equiv_2 1 \\ 2p+2q+8r-14 & p \equiv_2 1, q \equiv_2 1, r \equiv_2 0 \\ 2p+2q+8r-14 & p \equiv_2 1, q \equiv_2 1, r \equiv_2 1 \end{cases}$$

Proof: The state counts can be read off of the transition tables given in Appendix C. ■

Transition tables for word acceptors for (p,q,r)-triangle groups can be found in Appendix C. The 15 cases that are missing are the ones for $p < 6$, but these are easily computed as shown in this section. These cases have to be treated seperately because for $p < 6$ some parts of the rewrite rules are reduced to the empty string.

There are some quite interesting observerations that can be made here. First, if $[vx] = [wy]$ for representatives v and w of two distinct Nerode congruence classes, both not equal to the sink state, then $x = y$. This means that all incoming transitions to a state are labelled with the same generator.

For a generalisation we need a bit of vocabulary. Let $\mathfrak{W}$ be a word acceptor. We call a sequence $\mathcal{L} = q^{(1)}a_1q^{(2)}a_2\ldots a_{n-1}q^{(n)}$ of states a *loop labelled with* $v = a_1a_2\ldots a_n$ if $q^{(1)} = q^{(n)}$. We call the state $q^{(i)}$ whose minimal Nerode representative in some well order on the strings over the alphabet of $\mathfrak{W}$ is the minimal among all representatives of the states in the loop the *base* of the loop. We also say that the loop is based at $q^{(i)}$.

In the word acceptors for the triangle groups the following lemma holds.

Lemma 4.7 (loops):
Let $\mathfrak{W}$ be a word acceptor for a triangle group as computed above. Then for a loop $\mathcal{L}$ based at $[v] \in Q$ labelled with $w \in A^*$ the string w can be decomposed into $w = uv$.

Proof: This follows directly from the transition tables in Appendix C. ■

The bases of loops in word acceptors for triangle groups are the states $[x]$, $[y]$, $[X]$, $[Y]$, $[xx]$, $[xy]$, $[yx]$, $[yy]$, $[XX]$, $[XY]$, $[YX]$ and $[YY]$.

From a language theoretic standpoint the situation is easily explained: Because the language of the word acceptor contains exactly the strings that do not contain substrings that are left hand sides of rules, the acceptor has to count occurrences of powers of generators. Also, it seems that changing the set of generators does not change the structure of the acceptor too much. From the standpoint of group theory it is not clear if there is a connection between this fact and the structure of the group.

Also the above observations enable us to classify all elements of infinite order in the triangle groups and thus the subgroups isomorphic to $(\mathbb{Z},+)$. If a group element g has infinite order, there has to be an infinite sequence of strings of increasing length that represent the powers of g. By a pumping argument there can only be finitely many basic representatives of such elements. For the triangle groups, if p, q and r are large enough, for example every element $(xY)^k$ has infinite order because for every k the string $(xY)^k$ is accepted by the word acceptor.

As a last observation the language $L(\mathfrak{W})^{-1}$ is the same as $L(\mathfrak{W})$. This puts restrictions on the regular language that can be chosen for such an automatic presentation: A minimal deterministic finite state automaton may not have any blowup in states for the reversed language.

We will give generalised conjectures for the above observations in Chapter 5.

For triangle groups, a (p,q,r)-triangle group can be embedded into a (p',q',r')-triangle group if p divides p', q divides q' and r divides r' by an obvious mapping of the generators of the first group to powers of generators of the second. Also, if we have a word acceptor for the (p',q',r')-triangle group, we can find one for the (p,q,r)-triangle group by deleting states and interpreting powers of group generators as labels for transitions.

This leads up to the question whether we can find a kind of normal form for the automatic presentation of a triangle group. This would enable us to decide the isomorphism problem for the triangle groups. That is, given an automatic presentation of a group, does it present a triangle group with parameters (p,q,r). This seems rather restricted, nevertheless to the knowledge of the author it is currently unknown whether the isomorphism problem for automatic groups is decidable.

4.4 Changing Generators and Some Symmetries

It is unknown how exactly automatic presentations behave if we change the presentation of the group. For word-hyperbolic groups like the triangle groups the situation is a little bit better understood. Triangle groups have an easily understood automatic presentation for all sets of generators. We propose two group presentations that yield word acceptors that seem better suited for some arguments.

Further investigation should target how the word acceptors for triangle groups behave under Tietze transformations. Adding a relation that is already in the normal closure will not yield any change in the word acceptor, as these are either added or removed by the Knuth-Bendix completion process. This leaves adding or removing a generator a and adding a relation (a,w) where w is a string over the generating set. Some of the results in earlier sections suggest that we can then "rewrite" the word acceptor too resulting in a word acceptor over the new generating set.

The first presentation replaces xy by z as it was already described in Section 4.1 and thus we have the presentation

$$\mathrm{Mon}\left\langle\, x,y,z,X,Y,Z \mid R' \,\right\rangle$$

where

$$R' := \{(xX,\varepsilon),\ (Xx,\varepsilon),\ (yY,\varepsilon),\ (Yy,\varepsilon),\ (zZ,\varepsilon),\ (Zz,\varepsilon),\ (x^p,\varepsilon),\ (y^q,\varepsilon),\ (z^r,\varepsilon),\ (xyz,\varepsilon)\}.$$

The advantage of this presentation is that there is no need to count the $(xy)^k$ which adds complexity and states. We still count repetitions of x, y and z and their inverses. The following shows briefly how to construct a presentation that does not explicitly count occurrences.

We add for each generator u_i that has order greater than two a new generator $u_i^{(j)}$ and a relation $u_i' = u_i^2$. This is a Tietze transformation and thus we do not change the isomorphism type of the group. We can now replace the relation u_i^p by $u_i^{\lfloor \frac{p}{2} \rfloor}u$ where u is equal to one if p is even and $u = u_i$ if it is not. We can repeat this process until we reached a presentation of the triangle group that does not contain any generator of order greater than 2. This also means that in the language of the word acceptor there never is an infix of a string that is a power of a generator.

An interesting project for the future is exploring the properties of the aforementioned presentation.

Changing the ordering of the generators does not change the Cayley graph but only what is considered a **ShortLex** minimal representative. Thus the set of word differences does not change and subsequently the resulting word acceptors are isomorphic.

The set of rewrite rules becomes infinite as soon as a generator is directly adjacent to its inverse in the ordering. For example, the ordering $x < y < X < Y$ yields a finite set of rewrite rules but the ordering $x < X < y < Y$ yields an infinite one. This might be directly connected to the fact that $(xy)^r$ is a relation but the connection is not clear and needs further examination. Because the word acceptor is computed from the word difference automaton, the change here is expected to be minimal, although a formal proof of this fact is missing as well as the proper tools and notions to formalise this similarity.

4.5 Growth Functions

In this section we want to compute growth functions for the triangle groups. The methods used here are based on the paper [EIFZ96] where the computation of growth functions for automatic groups is described. We give a short overview of the procedure and then compute growth functions for triangle groups.

Suppose we have a **ShortLex** automatic presentation $\mathfrak{a} = \left(A, \pi, \mathfrak{W}, (\mathfrak{M}_x)_{x \in A \cup \varepsilon}\right)$ of a finiteley presented group $\mathfrak{G}$. We want to compute the sequence $(c_n)_{n \in \mathbb{N}}$ of natural numbers where c_n is the number of group elements that have a shortest representative over A of length precisely n. In the Cayley graph of $\mathfrak{G}$ with respect to the generating set A the number c_n is the number of elements that are at a distance exactly n from the identity vertex.

Assuming we know $(c_n)_{n \in \mathbb{N}}$, we can form the *growth function*

$$C_{(\mathfrak{G},A)}(z) = \sum_{n=0}^{\infty} c_n z^n,$$

by a formal power series. One can show that this is the power series of a holomorphic function and has radius of convergence of at least $(2|A|-1)^{-1}$ thus justifying the notion of a growth function instead of a formal growth series.

For a **ShortLex** automatic group it is possible to compute c_n for $n \in \mathbb{N}$ and it is even possible to compute a representation of the growth function $C_{(\mathfrak{G},A)}(z)$ as a quotient of two polynomials. We shortly describe how this is done.

Because for a **ShortLex** automatic group we have a word acceptor $\mathfrak{W}$ with a language that consists of unique shortest representatives for each group element, we merely need to count how many strings of length n are accepted by $\mathfrak{W}$. For this let Q be the set of states of $\mathfrak{W}$ with cardinality k. We number the states in Q by 1 to k and form the transition matrix $T := (t_{i,j})_{1 \leqslant i,j \leqslant k}$ where $t_{i,j}$ equals one if there is a transition (q_i, x, q_j) in the transition relation of $\mathfrak{W}$ for some $x \in A$ and zero otherwise.

A standard result in algebraic graph theory now states that we can compute the number of paths of length n in the transition graph of $\mathfrak{W}$ that start in a subset I of the set of vertices and ends in the subset F using the transition matrix.

Obviously we choose I to be the set that contains the inital state and F to be the set of final states of the automaton $\mathfrak{W}$. The *characteristic vector* $\chi(S)$ of a subset S of the set Q of states is an element of $\mathbb{Z}^k$ where $\chi(S)_i = 1$ if and only if q_i is an element of S. To compute the number of paths of length n

that originate in I and end in F, let $v = \chi(I)$ and $w = \chi(F)$. We can now compute $c_n = v^t T^n w$. Putting this into the growth series yields the following result:

$$\begin{aligned} \sum_{n=0}^{\infty} \left(v^t T^n w\right) z^n &= v^t \left(\sum_{n=0}^{\infty} (zT)^n \right) w \\ &= v^t \left(\mathrm{E}_k - zT\right)^{-1} w \\ &= \frac{P(z)}{Q(z)} \end{aligned}$$

Where E_k denotes the $k \times k$ identity matrix, $P(z)$ is a polynomial with integer coefficients of degree smaller than k and $Q(z) = \det(\mathrm{E}_k - zT)$. The last line follows by applying Cramer's rule to the matrix $(\mathrm{E}_k - zT)$. We also observe that $\det(\mathrm{E}_k - zT) = (-z)^n \det\left(T - z^{-1}\mathrm{E}_k\right)$.

To make computations possible we make the following key observation: The sequence $(c_n)_{n\in\mathbb{N}}$ satisfies a linear recurrence relation. That is, for some $m \in \mathbb{N}$ there are integers q_i such that

$$\sum_{i=0}^{m} q_i c_i = 0$$

To see this suppose $P(z) = \sum_{i=0}^{m-1} p_i z^i$ and $Q(z) = \sum_{i=0}^{m} q_i z^i$ to be polynomials in $\mathbb{Z}[z]$. Then by mere calculation

$$\begin{aligned} & & C(z) &= \tfrac{P(z)}{Q(z)} \\ &\Leftrightarrow & C(z)\,Q(z) &= P(z) \\ &\Leftrightarrow & \sum_{n=0}^{\infty} \left(\sum_{i=0}^{\min\{n,m\}} q_i c_{n-i} \right) z^n &= \sum_{i=0}^{m-1} p_i z^i \end{aligned}$$

Thus the coefficients p_i are already defined by the q_i and the first elements of $(c_n)_{n\in\mathbb{N}}$. The other way round

$$c_n = -\frac{1}{q_0} \sum_{i=1}^{m} q_i c_{n-i} \text{ for } n \geqslant m.$$

As a corollary, the sequence $(c_n)_{n\in\mathbb{N}}$ is for $n > k$ thus recursively defined in terms of $c_0, \ldots, c_k$.

We now want to compute growth functions for the triangle groups. Growth functions depend on the set of group generators, and we decide to use the presentation for which we computed the word acceptors earlier. As a small example we look at the group $\Delta(6,6,6)$. The **kbmag** package includes a program for computing growth functions of finite state acceptors, thus we can compute the growth function of $\Delta(6,6,6)$ for our presentation using this program.

$$C_{\Delta(6,6,6)}(z) = \frac{1 + 2z + 2z^2 + 2z^3 + 2z^4 + 2z^5 + z^6}{1 - 2z - 2z^2 - 2z^4 - 2z^5 + z^6}$$

We observe that the coefficients of the polynomials are all two or minus two except for the first and the last one which are both equal to one. This immedeately draws attention and after computing more growth functions we conjecture the following.

For $p = q = r$ and $p \equiv_2 0$ the growth function can be given as follows.

$$C_{\Delta(p,p,p)}(z) = \frac{1 + 2z + 2z^2 + \ldots + 2z^{p-1} + z^p}{1 - 2z - 2z^2 - \ldots - 2z^{\frac{p}{2}-1} - 2z^{\frac{p}{2}+1} - \ldots - 2z^{p-1} + z^p}.$$

Additionally, if we decompose the numerator of $C_{\Delta(6,6,6)}(z)$ into irreducible polynomials over $\mathbb{Q}$, then

$$1+2z+2z^2+2z^3+2z^4+2z^5+z^6 = (z+1)^2\left(z^2-z+1\right)\left(z^2+z+1\right)$$

Which are exactly the second, third and sixth cyclotomic polynomials. This is interesting because two, three and six are divisors of p and it might give a hint towards a closed formula for the growth functions of triangle groups. For the numerator of the growth function of such a triangle group the following seems to hold.

$$P(z) = \frac{(z^p-1)(z+1)}{(z-1)}$$

Some of these obvservations do not hold anymore if we look at general triangle groups. We observe that the polynomials in the numerator and the denominator are *palindromic*. We call a polynomial of degree n over a ring R with coefficients $a_0, a_1, \ldots, a_n$ *palindromic* if a_i is equal to a_{n-i} for $0 \leqslant i \leqslant n$.

Palindromic polynomials with real coefficients have the property that their roots occur in inverse pairs. If all roots are roots of unity, inversion coincides with complex conjugation, but this does not have to be the case. The tables in Appendix D show a number of examples of growth functions for triangle groups for varying p, q and r. There are automatic groups that do not have a growth function that has palindromic numerator and denominator.

Unfortunately there is no obvious way to give a closed form for the growth function of a triangle group and much less a direct proof of such a formula. The transition matrices for the triangle groups are quite easily described. Only the first fourteen rows have interesting entries. All further rows only have exactly one entry different from zero.

A promising approach seems to be looking at the Frobenius normal form of the transition matrix. This form is particularly easy to find for the transition matrices of word acceptors of triangle groups. In the Frobenius normal form of the matrix the top row resembles a linear recurrence relation for the growth function. It might not be the shortest but if we have a recurrence relation we can find a shortest one in there. Looking at the matrices in Appendix E, the Frobenius normal form seems quite easy to compute for the transition matrices. It was not possible until now to develop a closed form of a growth function for all triangle groups.

5 Towards some General Results

We want to investigate to what extent the observations in Section 4.3 carry over to more general classes of automatic groups. We mainly formulate some conjectures.

We let $\mathfrak{G} = \mathrm{Mon}\langle A \mid R \rangle$ be an automatic group generated by A. We assume A to be closed under inverses and that there are no redundant generators in A. We let $\mathfrak{W}$ be a unique word acceptor that resulted from the construction described in Chapter 3. We denote by W the language of $\mathfrak{W}$.

Conjecture 5.1 (transitions):
Let $[v]$ and $[w]$ be Nerode congruence classes of W and assume v and w to be **ShortLex** *minimal representatives. Let x and y be elements of A. If $[vx] = [wy]$ then $x = y$.* □

Conjecture 5.2 (loops):
For a loop $\mathcal{L}$ based at $[v] \in Q$ labelled with $w \in A^*$ the string w can be decomposed into $w = uv$. In particular w is itself an element of W. □

Sometimes we are interested in the order of group elements. We define the order of an element g of a group as the smallest natural number n such that $g^n = 1$. We say that g has infinite order, if there is no such n. For groups that allow for a unique word acceptor, we want to computationally find the languages of representatives of finite and infinite order by the use of the word acceptor.

A representative $v = a_1 a_2 a_3 \cdots a_n$ of a group element is called *cyclically reduced*, if every cyclic permutation of the string v is reduced. That is there is no cyclic permutation of v such there are adjacent representatives of group elements that are mutually inverse.

Conjecture 5.3 (infinite order):
Let $g \in \mathfrak{G}$ be an element of infinite order. If g has a cyclically reduced representative $v \in W$ then there is a $k \in \mathbb{N}$ such that v^k can be decomposed into two strings u and w and there is a loop based at u and labelled with wu. If g does not have a cyclically reduced representative in W then g is conjugate to an element h that has a cyclically reduced representative in W. □

If an element g not equal to the identity is represented by a string v for which v^i is contained in W, then g has infinite order. For this to hold the string v has to be cyclically reduced.

Remark 5.4 (infinite order):

- *If an element g of $\mathfrak{G}$ has a representative v such that v^i is contained in $L(\mathfrak{W})$ for all $i \in \mathbb{N}$, then g has infinite order.*
- *If there are strings u and v over A^* such that $[u]_{L(W)} = [uv]_{L(W)}$ and $v = xu$ then $ux = uvu^{-1}$ has infinite order. In particular v is cyclically reduced.*

Proof: Because v represents g, the string v^i represents g^i. If v is not equal to the empty string, v^i is also not equal to the empty string and because $\mathfrak{W}$ is unique, v^i does not represent the identity element of $\mathfrak{G}$.

Because $[u]_{L(\mathfrak{W})} = [uv]_{L(\mathfrak{W})}$, the string v labels a loop in $\mathfrak{W}$ and is thus irreducible and cyclically reduced. Additionally $(ux)^i$ is accepted by $\mathfrak{W}$ for all i, thus the claim holds.

■

The above remark captures only one direction of the supposed characterisation of elements of infinite order in an automatic group. We take a look at group elements of infinite order. Let g be an element of the group $\mathfrak{G}$ such that $g^n \neq 1$ for all $n \in \mathbb{N}$. Because $\mathfrak{W}$ is a unique word acceptor, there is a sequence of representatives v_n in $L(\mathfrak{W})$ of g^n for all $n \in \mathbb{N}$. We apply a pumping argument.

Lemma 5.5 (infinite order):
Let g be an element of $\mathfrak{G}$ of infinite order that has a cyclically reduced representative v. For some $k \in \mathbb{N}$ there is a representative v_k of g^k such that there is a decomposition of v_k into three strings $v_k = xyz$ and xy^iz is in $L(\mathfrak{W})$ and represents g^{ki}.

Proof: *Note this is only an incomplete sketch of a full proof.* Because g has infinite order and $\mathfrak{W}$ is a unique word acceptor, it is clear that there is a sequence v_k of unique representatives of g^k. As the length of these representatives has to increase, at some point the length of a representative of g^k for a k in $\mathbb{N}$ is greater than the count of states in the word acceptor. At this point we can apply a pumping argument and conclude that for $v_k = xyz$ also xy^iz is accepted by $\mathfrak{W}$. What we cannot conclude at this point is that xy^iz represents a power of g.

Because y labels a loop based at $[x]$, we can by Conjecture 5.3 conclude that $y = ux$.

Thus, $v_k = xuxz$ and $x(ux)^i z$ is accepted by the word acceptor. What is missing here is that $z = u$. If we also assume this to be true, because the string z has to be inside the loop which is completed by u, we can conclude the result.

Elements of infinite order in automatic groups could now be found and classified by inspecting the word acceptor. Thus, a part of the group structure is already found in the word acceptor. This is in contrast to Section 2.4 and needs investigation.

Also elements of finite order can be found, these are the representatives of elements such that there actually is a $k \in \mathbb{N}$ with v^k not in $L(\mathfrak{W})$. As a corollary, these elements have a very restricted structure, we can find representatives of them already in the finite word acceptor.

6 Conclusion and Further Work

"It's done, when it's done."

(english phrase)

As it is always the case in research the work is not done. The field of automatic presentations is a very busy research area. This thesis dealt with a small class of groups, the triangle groups. After a complete introduction to the theory, we described how we can find a finite set of rewrite rules for the triangle groups and resulting from this, we showed how word acceptors for triangle groups are formed.

There are strong hints that the word acceptors for triangle groups with respect to different generating sets and different orders on generating sets are in some sense similar. The proof of this fact stays open, mainly because first a good formalisation of this similarity has to be found.

We showed that there is a connection between the set of rewrite rules and the Nerode congurence classes of the word acceptor. After having computed the word acceptors we considered a natural application: Computing growth functions for triangle groups. This became possible because we have a nice description of a unique word acceptor for each triangle group.

The growth functions of triangle groups seem to have a representation as a quotient of two palindromic polynomials. That is, the sequence of coefficients is the same if it is read backwards. We were unable to prove this fact, but provided computational evidence that this is the case. A direct attack could involve showing that for each nonzero eigenvalue λ of the transition matrix, λ^{-1} also is an eigenvalue. More general results should investigate whether we can estimate the parameters p, q and r from a given growth function.

As a more general result we found out how to compute elements of infinite order in automatic groups. This might help in finding isomorphisms between automatic groups. It is interesting that there is a kind of duality between the Cayley graph of the group and the word acceptor. In the Cayley graph, a loop always means that an element has finite order, in the automaton a loop is always an indication for something that is infinite or at least arbitrarily long.

The next step should be finding the methods and proofs for the conjectures given in the last chapter. There are quite a few directions for further research.

Is it possible to give characterisations of groups that allow for a finite complete rewriting system or are rewriting systems not the tool of choice for algorithmic group theory? Is it possible do describe word acceptors for larger classes of automatic groups?

Additionally, can we find some kind of canonical presentation for triangle groups or even automatic groups and thus decide the isomorphism problem for automatic groups or are there subclasses of automatic groups as the triangle groups or the word hyperbolic groups for which we can? This last hypothesis is conjectured to be false by Epstein et al. in [EPC$^+$92]. However it might be possible to decide isomorphism at least for a subclass of automatic groups.

Can the automata in automatic presentations can be decomposed in a certain sense? Word acceptors of triangle groups contain certain characteristic loops that are dependent on the parameters p, q and r. Is it possible to find a general construction of word acceptors directly from the group presentation?

Can we give characteristics of group presentations that suggest that a group is automatic? It might be a good idea to construct non-deterministic finite state automata first.

Are there applications of ω-automata or tree automata in algebra or especially in automatic group theory? In triangle groups, the limit language of $L(\mathfrak{W})$ consists of infinite length geodesics, which seem to play a role a decision procedure for the conjugacy problem.

There is also a developed thory of so called pushdown systems that is connected to the theory of rewriting systems and context free grammars. It should thus be possible to give a characterisation of word hyperbolic groups, whose multiplication table is context free, by certain pushdown graphs.

Can we compute automatic structures for groups like the Heisenberg group that allow for an automatic presentation of its Cayley graph without being automatic in the classical sense? Or can we practically compute in some way automatic structures for groups that are automatic but not **ShortLex** automatic?

The syntactic semigroup of the language of a finite state automaton is generated by the state transformations a symbol in the alphabet induces on the minimal deterministic finite state automaton. Is it possible to find a natural connection between the structure of the group and the structure of such a transition semigroup? The author extracted generating matrices for the syntactic semigroup for the word language of the $(12,6,6)$-triangle group and did a few computational experiments with them. For example there are certain fixed point elements in the semigroup that are connected to loops in the word acceptor. We note that by Section 2.4 the connection between the syntactic semigroup of a word acceptor of a group an the group itself can only be limited, because there are groups that are not isomorphic but allow for equal word acceptors.

In conclusion there are many directions in which further research might go. Although we have an implemented algorithm and some strong theorems at our hands, there seem to be extensions as well as details that wait to be explored.

A Notation

Sets	A, B	latin capitals
Alphabet	A	finite set
Set of all strings over A	A^*	Set of all finite sequences of elements of A
Free group on A	$\mathcal{F}\ (A)$	
Free monoid on A	$\mathcal{M}\ (A)$	
Free semigroup on A	$\mathcal{S}\ (A)$	
Elements of A	a, b, c	
Elements of A^*	u, v, w, x, y, z	sequence of elements of A
Concatenation u, v	uv, $u \cdot v$	
Langugage	$L \subseteq A^*$	
Automaton	$\mathfrak{A}$, $\mathfrak{B}$	
Run of automaton on a string	$\mu(s)$	
Prefix relation	$u \prec v$, $u \preceq v$	
Prefix of length t	$w[t]$	
Suffix	$u \succ v$, $u \succeq v$	
Infix	$u \curlyvee v$	
Subword		
Group	$\mathfrak{G}$, $\mathfrak{H}$	
Maps	$f : M \to N$	latin characters
Maps (overset)	$M \xrightarrow{f} N$	
Morphisms	$\varphi : G \to H$	greek characters
	$\varphi_g : G \to H\ :\ x \mapsto xg$	
Algebraic Structures	$\mathfrak{A}$, $\mathfrak{B}$	
Group presentation with generators and relators	$\mathfrak{p} = \langle\ A\ \mid\ R\ \rangle$	
Automatic Presentations	$\mathfrak{a}$, $\mathfrak{b}$	
Cayley Graph of G w. resp to A	$\mathcal{G}\ (G, A)$	
Set of ShortLex representatives of elements of G with respect to A	$\mathrm{ShortLex}\,(G, A)$	
Closure	$\langle X \rangle$	
Normal Closure	$\langle\langle X \rangle\rangle$	

B Knuth-Bendix Completion in Presentations of the (p,q,r)-triangle Groups

For this we fix the set of monoid generators to be $x < y < X < Y$ in that order.

B.1 Inversion rules

For every generator x we have a formal inverse we denote by X. Therefore the following two rules are needed for every generator.

$$\begin{array}{lll} (i_{x,1}) & xX & \rightarrow & \varepsilon \\ (i_{x,2}) & Xx & \rightarrow & \varepsilon \end{array}$$

If we have a group presented as a monoid by a set A of generators and take the inversion rules for each generator in A as a set of rewrite rules, we get a complete set of rewrite rules for the free product of the infinite cyclic groups generated by the elements of A. In particular, if A only consists of one generator and its formal inverse, we have a finite complete rewriting system for a cyclic group, regardless of the order we impose on the generating set.

B.2 Cyclic groups

As stated above, a finite complete set of rewrite rules for infinite cyclic groups is easy to find. For finite cyclic groups this is as easy. We assume $x < X$. Letting the Symmetric group act on the generating set and on the set of rewriting rules leaves the set complete.

$$\begin{array}{ll} (p_{x,1}) & x^{\lceil \frac{p+1}{2} \rceil} \rightarrow X^{\lfloor \frac{p-1}{2} \rfloor} \\ (p_{x,2}) & X^{\lceil \frac{p}{2} \rceil} \rightarrow x^{\lfloor \frac{p}{2} \rfloor} \end{array}$$

B.3 The triangle groups

Because the triangle groups are a factor group of the free product of two cyclic groups we start with the following set of rewrite rules and add in the following section rewrite rules for $(xy)^r$.

$$\begin{array}{llllllll} (i_{x,1}) & xX & \rightarrow & \varepsilon & (i_{y,1}) & yY & \rightarrow & \varepsilon \\ (i_{x,2}) & Xx & \rightarrow & \varepsilon & (i_{y,2}) & Yy & \rightarrow & \varepsilon \\ (p_{x,1}) & x^{\lceil \frac{p+1}{2} \rceil} & \rightarrow & X^{\lfloor \frac{p-1}{2} \rfloor} & (p_{y,1}) & y^{\lceil \frac{q+1}{2} \rceil} & \rightarrow & Y^{\lfloor \frac{q-1}{2} \rfloor} \\ (p_{x,2}) & X^{\lceil \frac{p}{2} \rceil} & \rightarrow & x^{\lfloor \frac{p}{2} \rfloor} & (p_{y,2}) & Y^{\lceil \frac{q}{2} \rceil} & \rightarrow & y^{\lfloor \frac{q}{2} \rfloor} \end{array}$$

We first look at the two cases for r even and r odd and then give the finite complete sets of rewrite rules for all cases.

In the case that r is even, the Knuth-Bendix completion procedure produces the following rules when completing $(xy)^r \to \varepsilon$

$$\begin{array}{lllll}
(c_1) & (xy)^{\frac{r}{2}}x & \to & (YX)^{\frac{r}{2}-1}Y \\
(c_2) & (yx)^{\frac{r}{2}}y & \to & (XY)^{\frac{r}{2}-1}X \\
(c_3) & (YX)^{\frac{r}{2}} & \to & (xy)^{\frac{r}{2}} \\
(c_4) & (XY)^{\frac{r}{2}} & \to & (yx)^{\frac{r}{2}}
\end{array}$$

We look at overlaps with the already generated rules listed in Appendix B.2. Depending on p or q being even or odd the added rules might look different.

$$\begin{tikzcd}
 & (YX)^{\frac{r}{2}-1}Yx^{\lfloor\frac{p}{2}\rfloor} \arrow[dd, dotted, shift right=3, "p\equiv_2 0"'] \\
\underline{(xy)^{\frac{r}{2}}x}\,\underline{x^{\lceil\frac{p+1}{2}\rceil-1}} \arrow[ur, "c_1"] \arrow[dr, "p_{x,1}"'] & \\
 & (xy)^{\frac{r}{2}}X^{\lfloor\frac{p-1}{2}\rfloor} \arrow[uu, dotted, shift right=3, "p\equiv_2 1"']
\end{tikzcd}
\qquad
\begin{tikzcd}
 & (XY)^{\frac{r}{2}-1}Xy^{\lfloor\frac{q}{2}\rfloor} \arrow[dd, dotted, shift right=3, "q\equiv_2 0"'] \\
\underline{(yx)^{\frac{r}{2}}y}\,\underline{y^{\lceil\frac{q+1}{2}\rceil-1}} \arrow[ur, "c_2"] \arrow[dr, "p_{y,1}"'] & \\
 & (yx)^{\frac{r}{2}}Y^{\lfloor\frac{q-1}{2}\rfloor} \arrow[uu, dotted, shift right=3, "q\equiv_2 1"']
\end{tikzcd}$$

$$\begin{tikzcd}
 & X^{\lfloor\frac{p-1}{2}\rfloor}(yx)^{\frac{r}{2}} \arrow[dd, dotted, shift right=3, "p\equiv_2 0"'] \arrow[dd, dotted, shift left=3, "p\equiv_2 1"] \\
\underline{x^{\lceil\frac{p+1}{2}\rceil-1}x}\,\underline{(yx)^{\frac{r}{2}}} \arrow[ur, "p_{x,1}"] \arrow[dr, "c_1"'] & \\
 & x^{\lfloor\frac{p}{2}\rfloor}(YX)^{\frac{r}{2}-1}Y
\end{tikzcd}
\qquad
\begin{tikzcd}
 & Y^{\lfloor\frac{q-1}{2}\rfloor}(xy)^{\frac{r}{2}} \arrow[dd, dotted, shift right=3, "q\equiv_2 0"'] \arrow[dd, dotted, shift left=3, "q\equiv_2 2"] \\
\underline{y^{\lceil\frac{q+1}{2}\rceil-1}y}\,\underline{(xy)^{\frac{r}{2}}} \arrow[ur, "p_{y,1}"] \arrow[dr, "c_2"'] & \\
 & y^{\lceil\frac{q}{2}\rceil}(XY)^{\frac{r}{2}-1}X
\end{tikzcd}$$

$$\begin{tikzcd}
 & (xy)^{\frac{r}{2}}X^{\lfloor\frac{p-1}{2}\rfloor} \arrow[dd, dotted, shift left=3, "p\equiv_2 1"] \\
\underline{(YX)^{\frac{r}{2}-1}YX}\,\underline{X^{\lceil\frac{p}{2}\rceil-1}} \arrow[ur, "c_3"] \arrow[dr, "p_{x,2}"'] & \\
 & (YX)^{\frac{r}{2}-1}Yx^{\lfloor\frac{p}{2}\rfloor} \arrow[uu, dotted, shift left=3, "p\equiv_2 0"]
\end{tikzcd}
\qquad
\begin{tikzcd}
 & (yx)^{\frac{r}{2}}Y^{\lfloor\frac{q-1}{2}\rfloor} \arrow[dd, dotted, shift left=3, "q\equiv_2 1"] \\
\underline{(XY)^{\frac{r}{2}-1}XY}\,\underline{Y^{\lceil\frac{q}{2}\rceil-1}} \arrow[ur] \arrow[dr] & \\
 & (XY)^{\frac{r}{2}-1}Xy^{\lfloor\frac{q}{2}\rfloor} \arrow[uu, dotted, shift left=3, "q\equiv_2 0"]
\end{tikzcd}$$

$$\begin{tikzcd}
 & x^{\lfloor\frac{p}{2}\rfloor}Y(XY)^{\frac{r}{2}-1} \\
\underline{X^{\lceil\frac{p}{2}-1\rceil}X}\,\underline{Y(XY)^{\frac{r}{2}-1}} \arrow[ur, "p_{x,2}"] \arrow[dr, "c_4"'] & \\
 & X^{\lfloor\frac{p-1}{2}\rfloor}(yx)^{\frac{r}{2}} \arrow[uu, dotted, shift right=3, "p\equiv_2 0"] \arrow[uu, dotted, shift left=3, "p\equiv_2 1"']
\end{tikzcd}
\qquad
\begin{tikzcd}
 & y^{\lfloor\frac{q}{2}\rfloor}X(YX)^{\frac{r}{2}-1} \\
\underline{Y^{\lceil\frac{q}{2}-1\rceil}Y}\,\underline{X(YX)^{\frac{r}{2}-1}} \arrow[ur] \arrow[dr] & \\
 & Y^{\lfloor\frac{q-1}{2}\rfloor-1}(xy)^{\frac{r}{2}} \arrow[uu, dotted, shift right=3, "q\equiv_2 0"] \arrow[uu, dotted, shift left=3, "q\equiv_2 0"']
\end{tikzcd}$$

In the case that r is odd, the Knuth-Bendix completion procedure produces the following rules when completing $(xy)^r \to \varepsilon$

$$\begin{array}{llcl}
(1) & (xy)^{\frac{r+1}{2}} & \to & (YX)^{\frac{r-1}{2}} \\
(2) & (yx)^{\frac{r+1}{2}} & \to & (XY)^{\frac{r-1}{2}} \\
(3) & (YX)^{\frac{r-1}{2}}Y & \to & (xy)^{\frac{r-1}{2}}x \\
(4) & (XY)^{\frac{r-1}{2}}X & \to & (yx)^{\frac{r-1}{2}}y
\end{array}$$

We look at overlaps with the previously already generated rules.

$$\begin{tikzcd}
 & X^{\lfloor\frac{p-1}{2}\rfloor}y(xy)^{\frac{r-1}{2}} \arrow[dd, dotted, shift right=4, "p\equiv_2 0"'] \arrow[dd, dotted, shift left=4, "p\equiv_2 1"] \\
\underline{x^{\lceil\frac{p+1}{2}\rceil-1}\underline{xy(xy)^{\frac{r-1}{2}}}} \arrow[ur] \arrow[dr] & \\
 & x^{\lceil\frac{p-1}{2}\rceil}(YX)^{\frac{r-1}{2}}
\end{tikzcd}$$

$$\begin{tikzcd}
 & Y^{\lfloor\frac{q-1}{2}\rfloor}x(yx)^{\frac{r-1}{2}} \arrow[dd, dotted, shift right=4, "q\equiv_2 0"'] \arrow[dd, dotted, shift left=4, "q\equiv_2 1"] \\
\underline{y^{\lceil\frac{q+1}{2}\rceil-1}\underline{yx(yx)^{\frac{r-1}{2}}}} \arrow[ur] \arrow[dr] & \\
 & y^{\lceil\frac{q-1}{2}\rceil-1}(XY)^{\frac{r-1}{2}}
\end{tikzcd}$$

$$\begin{tikzcd}
 & (XY)^{\frac{r-1}{2}}x^{\lfloor\frac{p}{2}\rfloor} \arrow[dd, dotted, shift right=4, "p\equiv_2 0"'] \\
\underline{(yx)^{\frac{r-1}{2}}\underline{yxx^{\lceil\frac{p+1}{2}\rceil-1}}} \arrow[ur] \arrow[dr] & \\
 & (yx)^{\frac{r-1}{2}}yX^{\lfloor\frac{p-1}{2}\rfloor} \arrow[uu, dotted, shift right=4, "p\equiv_2 1"']
\end{tikzcd}$$

$$\begin{tikzcd}
 & (YX)^{\frac{r-1}{2}}y^{\lceil\frac{q-1}{2}\rceil} \arrow[dd, dotted, shift right=4, "q\equiv_2 0"'] \\
\underline{(xy)^{\frac{r-1}{2}}\underline{xyy^{\lceil\frac{q+1}{2}\rceil-1}}} \arrow[ur] \arrow[dr] & \\
 & (xy)^{\frac{r-1}{2}}xY^{\lfloor\frac{q-1}{2}\rfloor} \arrow[uu, dotted, shift right=4, "q\equiv_2 1"']
\end{tikzcd}$$

$$\begin{tikzcd}
 & (yx)^{\frac{r-1}{2}}yX^{\lfloor\frac{p-1}{2}\rfloor} \arrow[dd, dotted, shift left=4, "p\equiv_2 1"] \\
\underline{(XY)^{\frac{r-1}{2}}\underline{XX^{\lceil\frac{p}{2}\rceil-1}}} \arrow[ur] \arrow[dr] & \\
 & (XY)^{\frac{r-1}{2}}x^{\lfloor\frac{p}{2}\rfloor} \arrow[uu, dotted, shift left=4, "p\equiv_2 0"]
\end{tikzcd}$$

$$\begin{tikzcd}
 & (xy)^{\frac{r-1}{2}}xY^{\lceil\frac{q}{2}\rceil-1} \arrow[dd, dotted, shift left=4, "q\equiv_2 1"] \\
\underline{(YX)^{\frac{r-1}{2}}\underline{YY^{\lceil\frac{q}{2}\rceil-1}}} \arrow[ur] \arrow[dr] & \\
 & (YX)^{\frac{r-1}{2}}y^{\lfloor\frac{q}{2}\rfloor} \arrow[uu, dotted, shift left=4, "q\equiv_2 0"]
\end{tikzcd}$$

$$\begin{tikzcd}
 & x^{\lfloor\frac{p}{2}\rfloor}(YX)^{\frac{r-1}{2}} \\
\underline{X^{\lceil\frac{p}{2}\rceil-1}\underline{X(YX)^{\frac{r-1}{2}}}} \arrow[ur] \arrow[dr] & \\
 & X^{\lceil\frac{p-1}{2}\rceil}y(xy)^{\frac{r-1}{2}} \arrow[uu, dotted, shift right=4, "p\equiv_2 0"'] \arrow[uu, dotted, shift left=4, "p\equiv_2 1"]
\end{tikzcd}$$

$$\begin{tikzcd}
 & y^{\lfloor\frac{q}{2}\rfloor}(YX)^{\frac{r-1}{2}} \\
\underline{Y^{\lceil\frac{q}{2}\rceil-1}\underline{Y(YX)^{\frac{r-1}{2}}}} \arrow[ur] \arrow[dr] & \\
 & Y^{\lceil\frac{q}{2}\rceil-1}x(yx)^{\frac{r-1}{2}} \arrow[uu, dotted, shift right=4, "q\equiv_2 0"'] \arrow[uu, dotted, shift left=4, "q\equiv_2 1"]
\end{tikzcd}$$

C Rewriting Systems and Word Acceptors for (p,q,r)-triangle groups

In this section we give finite complete sets of rewrite rules for (p,q,r)-triangle groups and corresponding word acceptors as transition tables. The rest of this page is left blank because then the rewriting system and the automaton are printed next to each other.

C.1 $p \equiv_2 0$, $q \equiv_2 0$, $r \equiv_2 0$

1	xX	$\to$	ε
2	Xx	$\to$	ε
3	yY	$\to$	ε
4	Yy	$\to$	ε
5	$x^{\frac{p}{2}+1}$	$\to$	$X^{\frac{p}{2}-1}$
6	$X^{\frac{p}{2}}$	$\to$	$x^{\frac{p}{2}}$
7	$y^{\frac{q}{2}+1}$	$\to$	$Y^{\frac{q}{2}-1}$
8	$Y^{\frac{q}{2}}$	$\to$	$y^{\frac{q}{2}}$
9	$(xy)^{\frac{r}{2}}x$	$\to$	$(YX)^{\frac{r}{2}-1}Y$
10	$(yx)^{\frac{r}{2}}y$	$\to$	$(XY)^{\frac{r}{2}-1}X$
11	$(YX)^{\frac{r}{2}}$	$\to$	$(xy)^{\frac{r}{2}}$
12	$(XY)^{\frac{r}{2}}$	$\to$	$(yx)^{\frac{r}{2}}$
13	$(YX)^{\frac{r}{2}-1}Yx^{\frac{p}{2}}$	$\to$	$(xy)^{\frac{r}{2}}X^{\frac{p}{2}-1}$
14	$X^{\frac{p}{2}-1}(yx)^{\frac{r}{2}}$	$\to$	$x^{\frac{p}{2}}(YX)^{\frac{r}{2}-1}Y$
15	$(XY)^{\frac{r}{2}-1}Xy^{\frac{q}{2}}$	$\to$	$(yx)^{\frac{r}{2}}Y^{\frac{q}{2}-1}$
16	$Y^{\frac{q}{2}-1}(xy)^{\frac{r}{2}}$	$\to$	$y^{\frac{q}{2}}(XY)^{\frac{r}{2}-1}X$

State	x	y	X	Y	
$[]$	$[]$	$[]$	$[]$	$[]$	
$[\varepsilon]$	$[x]$	$[y]$	$[X]$	$[Y]$	
$[x^k]$	$[x^{k+1}]$	$[xy]$	$[]$	$[Y]$	$1 \leqslant k < \frac{p}{2}$
$[x^{\frac{p}{2}}]$	$[]$	$[xy]$	$[]$	$[Y]$	
$[y^k]$	$[yx]$	$[y^{k+1}]$	$[X]$	$[]$	$1 \leqslant k < \frac{q}{2}$
$[y^{\frac{q}{2}}]$	$[yx]$	$[]$	$[X]$	$[]$	
$[X^k]$	$[]$	$[y]$	$[X^{k+1}]$	$[XY]$	$1 \leqslant k < \frac{p}{2} - 1$
$[X^{\frac{p}{2}-1}]$	$[]$	$[xy]$	$[]$	$[XY]$	
$[Y^k]$	$[x]$	$[]$	$[YX]$	$[Y^{k+1}]$	$1 \leqslant k < \frac{q}{2} - 1$
$[Y^{\frac{q}{2}-1}]$	$[yx]$	$[]$	$[YX]$	$[]$	
$[(xy)^k]$	$[(xy)^k x]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r}{2}$
$[(xy)^{\frac{r}{2}}]$	$[]$	$[yy]$	$[X]$	$[]$	
$[(xy)^k x]$	$[xx]$	$[(xy)^{k+1}]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r}{2}$
$[(yx)^k]$	$[xx]$	$[(yx)^k y]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r}{2}$
$[(yx)^{\frac{r}{2}}]$	$[xx]$	$[]$	$[]$	$[Y]$	
$[(yx)^k y]$	$[(yx)^{k+1}]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(XY)^k]$	$[x]$	$[]$	$[(XY)^k X]$	$[YY]$	$1 \leqslant k < \frac{r}{2}$
$[(XY)^k X]$	$[]$	$[y]$	$[XX]$	$[(XY)^{k+1}]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(XY)^{\frac{r}{2}-1} X]$	$[]$	$[yy]$	$[XX]$	$[]$	
$[(YX)^k]$	$[]$	$[y]$	$[XX]$	$[(YX)^k Y]$	$1 \leqslant k < \frac{r}{2}$
$[(YX)^k Y]$	$[x]$	$[]$	$[YX]^{k+1}$	$[]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(YX)^{\frac{r}{2}-1} Y]$	$[xx]$	$[]$	$[]$	$[YY]$	

C.2 $p \equiv_2 0$, $q \equiv_2 0$, $r \equiv_2 1$

1	xX	$\rightarrow$	ε
2	Xx	$\rightarrow$	ε
3	yY	$\rightarrow$	ε
4	Yy	$\rightarrow$	ε
5	$x^{\frac{p}{2}+1}$	$\rightarrow$	$X^{\frac{p}{2}-1}$
6	$X^{\frac{p}{2}}$	$\rightarrow$	$x^{\frac{p}{2}}$
7	$y^{\frac{q}{2}+1}$	$\rightarrow$	$Y^{\frac{q}{2}-1}$
8	$Y^{\frac{q}{2}}$	$\rightarrow$	$y^{\frac{q}{2}}$
9	$(xy)^{\frac{r+1}{2}}$	$\rightarrow$	$(YX)^{\frac{r-1}{2}}$
10	$(yx)^{\frac{r+1}{2}}$	$\rightarrow$	$(XY)^{\frac{r-1}{2}}$
11	$(YX)^{\frac{r-1}{2}}Y$	$\rightarrow$	$(xy)^{\frac{r-1}{2}}x$
12	$(XY)^{\frac{r-1}{2}}X$	$\rightarrow$	$(yx)^{\frac{r-1}{2}}y$
13	$X^{\frac{p}{2}-1}y(xy)^{\frac{r-1}{2}}$	$\rightarrow$	$x^{\frac{p}{2}}(YX)^{\frac{r-1}{2}}$
14	$(XY)^{\frac{r-1}{2}}x^{\frac{p}{2}}$	$\rightarrow$	$(yx)^{\frac{r-1}{2}}yX^{\frac{p}{2}-1}$
15	$Y^{\frac{q}{2}-1}x(yx)^{\frac{r-1}{2}}$	$\rightarrow$	$y^{\frac{q}{2}}(XY)^{\frac{r-1}{2}}$
16	$(YX)^{\frac{r-1}{2}}y^{\frac{q}{2}}$	$\rightarrow$	$(xy)^{\frac{r-1}{2}}xY^{\frac{q}{2}-1}$

State	x	y	X	Y	
$[]$	$[]$	$[]$	$[]$	$[]$	
$[\varepsilon]$	$[x]$	$[y]$	$[X]$	$[Y]$	
$[x^k]$	$[x^{k+1}]$	$[xy]$	$[]$	$[Y]$	$1 \leqslant k < \frac{p}{2}$
$[x^{\frac{p}{2}}]$	$[]$	$[xy]$	$[]$	$[Y]$	
$[y^k]$	$[yx]$	$[y^{k+1}]$	$[X]$	$[]$	$1 \leqslant k < \frac{q}{2}$
$[y^{\frac{q}{2}}]$	$[yx]$	$[]$	$[X]$	$[]$	
$[X^k]$	$[]$	$[y]$	$[X^{k+1}]$	$[XY]$	$1 \leqslant k < \frac{p}{2} - 1$
$[X^{\frac{p}{2}-1}]$	$[]$	$[xy]$	$[]$	$[XY]$	
$[Y^k]$	$[x]$	$[]$	$[YX]$	$[Y^{k+1}]$	$1 \leqslant k < \frac{q}{2} - 1$
$[Y^{\frac{q}{2}-1}]$	$[yx]$	$[]$	$[YX]$	$[]$	
$[(xy)^k]$	$[(xy)^k x]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k \leqslant \frac{r-1}{2}$
$[(xy)^k x]$	$[xx]$	$[(xy)^{k+1}]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r-1}{2}$
$[(xy)^{\frac{r-1}{2}} x]$	$[xx]$	$[]$	$[]$	$[Y]$	
$[(yx)^k]$	$[xx]$	$[(yx)^k y]$	$[]$	$[Y]$	$1 \leqslant k \leqslant \frac{r-1}{2}$
$[(yx)^k y]$	$[(yx)^{k+1}]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r-1}{2}$
$[(yx)^{\frac{r-1}{2}} y]$	$[]$	$[yy]$	$[X]$	$[]$	
$[(XY)^k]$	$[x]$	$[]$	$[(XY)^k X]$	$[YY]$	$1 \leqslant k < \frac{r-1}{2}$
$[(XY)^{\frac{r-1}{2}}]$	$[xx]$	$[]$	$[]$	$[YY]$	
$[(XY)^k X]$	$[]$	$[y]$	$[XX]$	$[(XY)^{k+1}]$	$1 \leqslant k < \frac{r-1}{2}$
$[(YX)^k]$	$[]$	$[y]$	$[XX]$	$[(YX)^k Y]$	$1 \leqslant k < \frac{r-1}{2}$
$[(YX)^{\frac{r-1}{2}}]$	$[]$	$[yy]$	$[XX]$	$[]$	
$[(YX)^k Y]$	$[x]$	$[]$	$[(YX)^{k+1}]$	$[YY]$	$1 \leqslant k < \frac{r-1}{2}$

C.3 $p \equiv_2 1$, $q \equiv_2 0$, $r \equiv_2 0$

1	xX	$\rightarrow$	ε
2	Xx	$\rightarrow$	ε
3	yY	$\rightarrow$	ε
4	Yy	$\rightarrow$	ε
5	$x^{\frac{p+1}{2}}$	$\rightarrow$	$X^{\frac{p-1}{2}}$
6	$X^{\frac{p+1}{2}}$	$\rightarrow$	$x^{\frac{p-1}{2}}$
7	$y^{\frac{q}{2}+1}$	$\rightarrow$	$Y^{\frac{q}{2}-1}$
8	$Y^{\frac{q}{2}+1}$	$\rightarrow$	$y^{\frac{q}{2}}$
9	$(xy)^{\frac{r}{2}}x$	$\rightarrow$	$(YX)^{\frac{r}{2}-1}Y$
10	$(yx)^{\frac{r}{2}}y$	$\rightarrow$	$(XY)^{\frac{r}{2}-1}X$
11	$(YX)^{\frac{r}{2}}$	$\rightarrow$	$(xy)^{\frac{r}{2}}$
12	$(XY)^{\frac{r}{2}}$	$\rightarrow$	$(yx)^{\frac{r}{2}}$
13	$(xy)^{\frac{r}{2}}X^{\frac{p-1}{2}}$	$\rightarrow$	$(YX)^{\frac{r}{2}-1}Yx^{\frac{p-1}{2}}$
14	$X^{\frac{p-1}{2}}(yx)^{\frac{r}{2}}$	$\rightarrow$	$x^{\frac{p-1}{2}}(YX)^{\frac{r}{2}-1}Y$
15	$(XY)^{\frac{r}{2}-1}Xy^{\frac{q}{2}}$	$\rightarrow$	$(yx)^{\frac{r}{2}}Y^{\frac{q}{2}-1}$
16	$Y^{\frac{q}{2}-1}(xy)^{\frac{r}{2}}$	$\rightarrow$	$y^{\frac{q}{2}}(XY)^{\frac{r}{2}-1}X$
17	$X^{\frac{p-1}{2}}y(xy)^{\frac{r}{2}-1}X^{\frac{p-1}{2}}$	$\rightarrow$	$x^{\frac{p-1}{2}}(YX)^{\frac{r}{2}-1}Yx^{\frac{p-1}{2}}$
18	$(xy)^{\frac{r}{2}}X^{\frac{p-3}{2}}(yx)^{\frac{r}{2}}$	$\rightarrow$	$(YX)^{\frac{r}{2}-1}Yx^{\frac{p-1}{2}}Y(XY)^{\frac{r}{2}-1}$
19	$Y(XY)^{\frac{r}{2}-1}x^{\frac{p-1}{2}}Y(XY)^{\frac{r}{2}-1}x^{\frac{p-1}{2}}$	$\rightarrow$	$(xy)^{\frac{r}{2}}X^{\frac{p-3}{2}}y(xy)^{\frac{r}{2}-1}X^{\frac{p-1}{2}}$
20	$X^{\frac{p-1}{2}}y(xy)^{\frac{r}{2}-1}X^{\frac{p-3}{2}}(yx)^{\frac{r}{2}}$	$\rightarrow$	$x^{\frac{p-1}{2}}Y(XY)^{\frac{r}{2}-1}x^{\frac{p-1}{2}}Y(XY)^{\frac{r}{2}-1}$

State	x	y	X	Y	
$[]$	$[]$	$[]$	$[]$	$[]$	
$[\varepsilon]$	$[x]$	$[y]$	$[X]$	$[Y]$	
$[x^k]$	$[x^{k+1}]$	$[xy]$	$[]$	$[Y]$	$1 \leqslant k < \frac{p-1}{2}$
$[x^{\frac{p-1}{2}}]$	$[]$	$[xy]$	$[]$	$[Y]$	
$[y^k]$	$[yx]$	$[y^{k+1}]$	$[X]$	$[]$	$1 \leqslant k < \frac{q}{2}$
$[y^{\frac{q}{2}}]$	$[yx]$	$[]$	$[X]$	$[]$	
$[X^k]$	$[]$	$[y]$	$[X^{k+1}]$	$[XY]$	$1 \leqslant k < \frac{p-1}{2}$
$[X^{\frac{p-1}{2}}]$	$[]$	$[xy]$	$[]$	$[XY]$	
$[Y^k]$	$[x]$	$[]$	$[YX]$	$[Y^{k+1}]$	$1 \leqslant k < \frac{q}{2} - 1$
$[Y^{\frac{q}{2}-1}]$	$[yx]$	$[]$	$[YX]$	$[]$	
$[(xy)^k]$	$[(xy)^k x]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r}{2}$
$[(xy)^k x]$	$[xx]$	$[(xy)^{k+1}]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r}{2}$
$[(xy)^{\frac{r}{2}}]$	$[]$	$[yy]$	$[(xy)^{\frac{r}{2}} X]$	$[]$	
$[(xy)^{\frac{r}{2}} X^k]$	$[]$	$[y]$	$[(xy)^{\frac{r}{2}} X^{k+1}]$	$[XY]$	$1 \leqslant k < \frac{p-3}{2}$
$[(xy)^{\frac{r}{2}} X^{\frac{p-3}{2}}]$	$[]$	$[(xy)^{\frac{r}{2}} X^{\frac{p-3}{2}} y]$	$[]$	$[XY]$	
$[(xy)^{\frac{r}{2}} X^{\frac{p-3}{2}} (yx)^k y]$	$[(xy)^{\frac{r}{2}} X^{\frac{p-3}{2}} (yx)^{k+1}]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(xy)^{\frac{r}{2}} X^{\frac{p-3}{2}} (yx)^k]$	$[xx]$	$[(xy)^{\frac{r}{2}} X^{\frac{p}{2}} (yx)^k y]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(xy)^{\frac{r}{2}} X^{\frac{p-3}{2}} (yx)^{\frac{r}{2}-1} y]$	$[]$	$[yy]$	$[X]$	$[]$	
$[(yx)^k]$	$[xx]$	$[(yx)^k y]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r}{2}$
$[(yx)^{\frac{r}{2}}]$	$[xx]$	$[]$	$[]$	$[Y]$	
$[(yx)^k y]$	$[(yx)^{k+1}]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(XY)^k]$	$[x]$	$[]$	$[(XY)^k X]$	$[YY]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(XY)^k X]$	$[]$	$[y]$	$[XX]$	$[(XY)^{k+1}]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(XY)^{\frac{r}{2}-1} X]$	$[]$	$[yy]$	$[XX]$	$[]$	
$[(YX)^k]$	$[]$	$[y]$	$[XX]$	$[(YX)^k Y]$	$1 \leqslant k < \frac{r}{2}$
$[(YX)^k Y]$	$[x]$	$[]$	$[(YX)^{k+1}]$	$[YY]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(YX)^{\frac{r}{2}-1} Y]$	$[(YX)^{\frac{r}{2}-1} Yx]$	$[]$	$[]$	$[YY]$	
$[(YX)^{\frac{r}{2}-1} Yx^k]$	$[(YX)^{\frac{r}{2}-1} Yx^{k+1}]$	$[xy]$	$[]$	$[Y]$	$1 \leqslant k < \frac{p-1}{2}$
$[(YX)^{\frac{r}{2}-1} Yx^{\frac{p-1}{2}}]$	$[]$	$[xy]$	$[]$	$[(YX)^{\frac{r}{2}-1} Yx^{\frac{p-1}{2}} Y]$	
$[(YX)^{\frac{r}{2}-1} Yx^{\frac{p-1}{2}} (YX)^k Y]$	$[x]$	$[]$	$[(YX)^{\frac{r}{2}-1} Yx^{\frac{p-1}{2}} (YX)^{k+1}]$	$[YY]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(YX)^{\frac{r}{2}-1} Yx^{\frac{p-1}{2}} (YX)^k]$	$[]$	$[y]$	$[XX]$	$[YY]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(YX)^{\frac{r}{2}-1} Yx^{\frac{p-1}{2}} (YX)^{\frac{r}{2}-1} Y]$	$[xx]$	$[]$	$[]$	$[YY]$	

C.4 $p \equiv_2 0$, $q \equiv_2 1$, $r \equiv_2 0$

1	xX	$\rightarrow$	ε
2	Xx	$\rightarrow$	ε
3	yY	$\rightarrow$	ε
4	Yy	$\rightarrow$	ε
5	$x^{\frac{p}{2}+1}$	$\rightarrow$	$X^{\frac{p}{2}-1}$
6	$X^{\frac{p}{2}}$	$\rightarrow$	$x^{\frac{p}{2}}$
7	$y^{\frac{q+1}{2}}$	$\rightarrow$	$Y^{\frac{q-1}{2}}$
8	$Y^{\frac{q+1}{2}}$	$\rightarrow$	$y^{\frac{q-1}{2}}$
9	$(xy)^{\frac{r}{2}}x$	$\rightarrow$	$(YX)^{\frac{r}{2}-1}Y$
10	$(yx)^{\frac{r}{2}}y$	$\rightarrow$	$(XY)^{\frac{r}{2}-1}X$
11	$(YX)^{\frac{r}{2}}$	$\rightarrow$	$(xy)^{\frac{r}{2}}$
12	$(XY)^{\frac{r}{2}}$	$\rightarrow$	$(yx)^{\frac{r}{2}}$
13	$(YX)^{\frac{r}{2}-1}Yx^{\frac{p}{2}}$	$\rightarrow$	$(xy)^{\frac{r}{2}}X^{\frac{p-3}{2}}$
14	$X^{\frac{p}{2}-1}(yx)^{\frac{r}{2}}$	$\rightarrow$	$x^{\frac{p}{2}}(YX)^{\frac{r}{2}-1}Y$
15	$(yx)^{\frac{r}{2}}Y^{\frac{q-1}{2}}$	$\rightarrow$	$(XY)^{\frac{r}{2}-1}Xy^{\frac{q-1}{2}}$
16	$Y^{\frac{q-1}{2}}(xy)^{\frac{r}{2}}$	$\rightarrow$	$y^{\frac{q-1}{2}}(XY)^{\frac{r}{2}-1}X$
17	$Y^{\frac{q-1}{2}}x(yx)^{\frac{r}{2}-1}Y^{\frac{q-1}{2}}$	$\rightarrow$	$y^{\frac{q-1}{2}}(XY)^{\frac{r}{2}-1}Xy^{\frac{q-1}{2}}$
18	$(yx)^{\frac{r}{2}}Y^{\frac{q-3}{2}}(xy)^{\frac{r}{2}}$	$\rightarrow$	$(XY)^{\frac{r}{2}-1}Xy^{\frac{q-1}{2}}X(YX)^{\frac{r}{2}-1}$
19	$(XY)^{\frac{r}{2}-1}Xy^{\frac{q-1}{2}}X(YX)^{\frac{r}{2}-1}y^{\frac{q-1}{2}}$	$\rightarrow$	$(yx)^{\frac{r}{2}}Y^{\frac{q-3}{2}}x(yx)^{\frac{r}{2}-1}y^{\frac{q-1}{2}}$
20	$Y^{\frac{q-1}{2}}x(yx)^{\frac{r}{2}-1}Y^{\frac{q-3}{2}}(xy)^{\frac{r}{2}}$	$\rightarrow$	$y^{\frac{q-1}{2}}X(YX)^{\frac{r}{2}-1}y^{\frac{q-1}{2}}X(YX)^{\frac{r}{2}-1}$

State	x	y	X	Y	
$[]$	$[]$	$[]$	$[]$	$[]$	
$[\varepsilon]$	$[x]$	$[y]$	$[X]$	$[Y]$	
$[x^k]$	$[x^{k+1}]$	$[xy]$	$[]$	$[Y]$	$1 \leqslant k < \frac{p}{2}$
$[x^{\frac{p}{2}}]$	$[]$	$[xy]$	$[]$	$[Y]$	
$[y^k]$	$[yx]$	$[y^{k+1}]$	$[X]$	$[]$	$1 \leqslant k < \frac{q-1}{2}$
$[y^{\frac{q-1}{2}}]$	$[yx]$	$[]$	$[X]$	$[]$	
$[X^k]$	$[]$	$[y]$	$[X^{k+1}]$	$[XY]$	$1 \leqslant k < \frac{p}{2} - 1$
$[X^{\frac{p}{2}-1}]$	$[]$	$[xy]$	$[]$	$[XY]$	
$[Y^k]$	$[x]$	$[]$	$[YX]$	$[Y^{k+1}]$	$1 \leqslant k < \frac{q-1}{2}$
$[Y^{\frac{q-1}{2}}]$	$[yx]$	$[]$	$[YX]$	$[]$	
$[(xy)^k]$	$[(xy)^k x]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r}{2}$
$[(xy)^k x]$	$[xx]$	$[(xy)^{k+1}]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r}{2}$
$[(xy)^{\frac{r}{2}}]$	$[]$	$[yy]$	$[X]$	$[]$	
$[(yx)^k]$	$[xx]$	$[(yx)^k y]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r}{2}$
$[(yx)^k y]$	$[(yx)^{k+1}]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(yx)^{\frac{r}{2}}]$	$[xx]$	$[]$	$[]$	$[(yx)^{\frac{r}{2}} Y]$	
$[(yx)^{\frac{r}{2}} Y^k]$	$[x]$	$[]$	$[YX]$	$[(yx)^{\frac{r}{2}} Y^{k+1}]$	$1 \leqslant k < \frac{q-3}{2}$
$[(yx)^{\frac{r}{2}} Y^{\frac{q-3}{2}}]$	$[(yx)^{\frac{r}{2}} Y^{\frac{q-3}{2}} x]$	$[]$	$[YX]$	$[]$	
$[(yx)^{\frac{r}{2}} Y^{\frac{q-3}{2}} (xy)^k x]$	$[xx]$	$[(yx)^{\frac{r}{2}} Y^{\frac{q-3}{2}} (xy)^{k+1}]$	$[]$	$[Y]$	$0 \leqslant k < \frac{r}{2} - 1$
$[(yx)^{\frac{r}{2}} Y^{\frac{q-3}{2}} (xy)^k]$	$[(yx)^{\frac{r}{2}} Y^{\frac{q-3}{2}} (xy)^k x]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(yx)^{\frac{r}{2}} Y^{\frac{q-3}{2}} (xy)^{\frac{r}{2}-1}]$	$[xx]$	$[]$	$[]$	$[Y]$	
$[(XY)^k]$	$[x]$	$[]$	$[(XY)^k X]$	$[YY]$	$1 \leqslant k < \frac{r}{2}$
$[(XY)^k X]$	$[]$	$[y]$	$[XX]$	$[(XY)^{k+1}]$	$1 \leqslant k < \frac{r}{2}$
$[(XY)^{\frac{r}{2}-1} X]$	$[]$	$[(XY)^{\frac{r}{2}-1} Xy]$	$[XX]$	$[]$	
$[(XY)^{\frac{r}{2}-1} Xy^k]$	$[yx]$	$[(XY)^{\frac{r}{2}-1} Xy^{k+1}]$	$[X]$	$[]$	$1 \leqslant k < \frac{q-1}{2}$
$[(XY)^{\frac{r}{2}-1} Xy^{\frac{q-1}{2}}]$	$[yx]$	$[]$	$[(XY)^{\frac{r}{2}-1} Xy^{\frac{q-1}{2}} X]$	$[]$	$1 \leqslant k < \frac{q-1}{2}$
$[(XY)^{\frac{r}{2}-1} Xy^{\frac{q-1}{2}} (XY)^k X]$	$[]$	$[y]$	$[XX]$	$[(XY)^{\frac{r}{2}-1} Xy^{\frac{q-1}{2}} (XY)^{k+1}]$	$0 \leqslant k < \frac{r}{2} - 1$
$[(XY)^{\frac{r}{2}-1} Xy^{\frac{q-1}{2}} (XY)^k]$	$[x]$	$[]$	$[(XY)^{\frac{r}{2}-1} Xy^{\frac{q-1}{2}} (XY)^k X]$	$[YY]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(XY)^{\frac{r}{2}-1} Xy^{\frac{q-1}{2}} (XY)^{\frac{r}{2}-1} X]$	$[]$	$[yy]$	$[XX]$	$[]$	
$[(YX)^k]$	$[]$	$[y]$	$[XX]$	$[(YX)^k Y]$	$1 \leqslant k < \frac{r}{2}$
$[(YX)^k Y]$	$[x]$	$[]$	$[(YX)^{k+1}]$	$[YY]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(YX)^{\frac{r}{2}-1} Y]$	$[xx]$	$[]$	$[]$	$[YY]$	

C.5 $p \equiv_2 1$, $q \equiv_2 1$, $r \equiv_2 0$

1	xX	$\rightarrow$	ε
2	Xx	$\rightarrow$	ε
3	yY	$\rightarrow$	ε
4	Yy	$\rightarrow$	ε
5	$x^{\frac{p+1}{2}}$	$\rightarrow$	$X^{\frac{p-1}{2}}$
6	$X^{\frac{p+1}{2}}$	$\rightarrow$	$x^{\frac{p-1}{2}}$
7	$y^{\frac{q+1}{2}}$	$\rightarrow$	$Y^{\frac{q-1}{2}}$
8	$Y^{\frac{q+1}{2}}$	$\rightarrow$	$y^{\frac{q-1}{2}}$
9	$(xy)^{\frac{r}{2}}x$	$\rightarrow$	$(YX)^{\frac{r}{2}-1}Y$
10	$(yx)^{\frac{r}{2}}y$	$\rightarrow$	$(XY)^{\frac{r}{2}-1}X$
11	$(YX)^{\frac{r}{2}}$	$\rightarrow$	$(xy)^{\frac{r}{2}}$
12	$(XY)^{\frac{r}{2}}$	$\rightarrow$	$(yx)^{\frac{r}{2}}$
13	$(xy)^{\frac{r}{2}}X^{\frac{p-1}{2}}$	$\rightarrow$	$(YX)^{\frac{r}{2}-1}Yx^{\frac{p-1}{2}}$
14	$X^{\frac{p-1}{2}}(yx)^{\frac{r}{2}}$	$\rightarrow$	$x^{\frac{p-1}{2}}(YX)^{\frac{r}{2}-1}Y$
15	$(yx)^{\frac{r}{2}}Y^{\frac{q-1}{2}}$	$\rightarrow$	$(XY)^{\frac{r}{2}-1}Xy^{\frac{q-1}{2}}$
16	$Y^{\frac{q-1}{2}}(xy)^{\frac{r}{2}}$	$\rightarrow$	$y^{\frac{q+1}{2}}(XY)^{\frac{r}{2}-1}X$
17	$X^{\frac{p-1}{2}}y(xy)^{\frac{r}{2}-1}X^{\frac{p-1}{2}}$	$\rightarrow$	$x^{\frac{p-1}{2}}(YX)^{\frac{r}{2}-1}Yx^{\frac{p-1}{2}}$
18	$(xy)^{\frac{r}{2}}X^{\frac{p-3}{2}}(yx)^{\frac{r}{2}}$	$\rightarrow$	$(YX)^{\frac{r}{2}-1}Yx^{\frac{p-1}{2}}Y(XY)^{\frac{r}{2}-1}$
19	$(YX)^{\frac{r}{2}-1}Yx^{\frac{p-1}{2}}Y(XY)^{\frac{r}{2}-1}x^{\frac{p-1}{2}}$	$\rightarrow$	$(xy)^{\frac{r}{2}}X^{\frac{p-3}{2}}y(xy)^{\frac{r}{2}-1}X^{\frac{p-1}{2}}$
20	$X^{\frac{p-1}{2}}y(xy)^{\frac{r}{2}-1}X^{\frac{p-3}{2}}(yx)^{\frac{r}{2}}$	$\rightarrow$	$x^{\frac{p-1}{2}}Y(XY)^{\frac{r}{2}-1}x^{\frac{p-1}{2}}Y(XY)^{\frac{r}{2}-1}$
21	$Y^{\frac{q-1}{2}}x(yx)^{\frac{r}{2}-1}Y^{\frac{q-1}{2}}$	$\rightarrow$	$y^{\frac{q-1}{2}}(XY)^{\frac{r}{2}-1}Xy^{\frac{q-1}{2}}$
22	$(yx)^{\frac{r}{2}}Y^{\frac{q-3}{2}}(xy)^{\frac{r}{2}}$	$\rightarrow$	$(XY)^{\frac{r}{2}-1}Xy^{\frac{q-1}{2}}X(YX)^{\frac{r}{2}-1}$
23	$(XY)^{\frac{r}{2}-1}Xy^{\frac{q-1}{2}}X(YX)^{\frac{r}{2}-1}y^{\frac{q-1}{2}}$	$\rightarrow$	$(yx)^{\frac{r}{2}}Y^{\frac{q-3}{2}}x(yx)^{\frac{r}{2}-1}Y^{\frac{q-1}{2}}$
24	$Y^{\frac{q-1}{2}}x(yx)^{\frac{r}{2}-1}Y^{\frac{q-3}{2}}(xy)^{\frac{r}{2}}$	$\rightarrow$	$y^{\frac{q-1}{2}}X(YX)^{\frac{r}{2}-1}y^{\frac{q-1}{2}}X(YX)^{\frac{r}{2}-1}$

State	x	y	X	Y	
$[]$	$[]$	$[]$	$[]$	$[]$	
$[\varepsilon]$	$[x]$	$[y]$	$[X]$	$[Y]$	
$[x^k]$	$[x^{k+1}]$	$[xy]$	$[]$	$[Y]$	$1 \leqslant k < \frac{p-1}{2}$
$[x^{\frac{p-1}{2}}]$	$[]$	$[xy]$	$[]$	$[Y]$	
$[y^k]$	$[yx]$	$[y^{k+1}]$	$[X]$	$[]$	$1 \leqslant k < \frac{q-1}{2}$
$[y^{\frac{q-1}{2}}]$	$[yx]$	$[]$	$[X]$	$[]$	
$[X^k]$	$[]$	$[y]$	$[X^{k+1}]$	$[XY]$	$1 \leqslant k < \frac{p-1}{2}$
$[X^{\frac{p-1}{2}}]$	$[]$	$[xy]$	$[]$	$[XY]$	
$[Y^k]$	$[x]$	$[]$	$[YX]$	$[Y^{k+1}]$	$1 \leqslant k < \frac{q-1}{2}$
$[Y^{\frac{q-1}{2}}]$	$[yx]$	$[]$	$[YX]$	$[]$	
$[(xy)^k]$	$[(xy)^k x]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r}{2}$
$[(xy)^k x]$	$[xx]$	$[(xy)^{k+1}]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r}{2}$
$[(xy)^{\frac{r}{2}}]$	$[]$	$[yy]$	$[(xy)^{\frac{r}{2}} X]$	$[]$	
$[(xy)^{\frac{r}{2}} X^k]$	$[]$	$[y]$	$[(xy)^{\frac{r}{2}} X^{k+1}]$	$[XY]$	$1 \leqslant k < \frac{p-3}{2}$
$[(xy)^{\frac{r}{2}} X^{\frac{p-3}{2}}]$	$[]$	$[(xy)^{\frac{r}{2}} X^{\frac{p-3}{2}} y]$	$[]$	$[XY]$	
$[(xy)^{\frac{r}{2}} X^{\frac{p-3}{2}} (yx)^k]$	$[]$	$[(xy)^{\frac{r}{2}} X^{\frac{p-3}{2}} (yx)^k y]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(xy)^{\frac{r}{2}} X^{\frac{p-3}{2}} (yx)^k y]$	$[(xy)^{\frac{r}{2}} X^{\frac{p-3}{2}} (yx)^{k+1}]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(xy)^{\frac{r}{2}} X^{\frac{p-3}{2}} (yx)^{\frac{r}{2}-1} y]$	$[]$	$[yy]$	$[X]$	$[]$	
$[(yx)^k]$	$[xx]$	$[(yx)^k y]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r}{2}$
$[(yx)^k y]$	$[(yx)^{k+1}]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(yx)^{\frac{r}{2}}]$	$[xx]$	$[]$	$[]$	$[(yx)^{\frac{r}{2}} Y]$	
$[(yx)^{\frac{r}{2}} Y^k]$	$[x]$	$[]$	$[YX]$	$[(yx)^{\frac{r}{2}} Y^{k+1}]$	$1 \leqslant k < \frac{q-3}{2}$
$[(yx)^{\frac{r}{2}} Y^{\frac{q-3}{2}}]$	$[(yx)^{\frac{r}{2}} Y^{\frac{q-3}{2}} x]$	$[]$	$[YX]$	$[]$	
$[(yx)^{\frac{r}{2}} Y^{\frac{q-3}{2}} (xy)^k]$	$[(yx)^{\frac{r}{2}} Y^{\frac{r-3}{2}} (xy)^k x]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r-1}{2}$
$[(yx)^{\frac{r}{2}} Y^{\frac{q-3}{2}} (xy)^k x]$	$[xx]$	$[(yx)^{\frac{r}{2}} Y^{\frac{r-3}{2}} (xy)^{k+1}]$	$[]$	$[Y]$	$0 \leqslant k < \frac{r-1}{2}$
$[(yx)^{\frac{r}{2}} Y^{\frac{q-3}{2}} (xy)^{\frac{r}{2}-1} x]$	$[xx]$	$[]$	$[]$	$[Y]$	
$[(XY)^k]$	$[x]$	$[]$	$[(XY)^k X]$	$[YY]$	$1 \leqslant k < \frac{r}{2}$
$[(XY)^k X]$	$[]$	$[y]$	$[XX]$	$[(XY)^{k+1}]$	$1 \leqslant k < \frac{r}{2}$
$[(XY)^{\frac{r}{2}-1} X]$	$[]$	$[(XY)^{\frac{r}{2}-1} Xy]$	$[XX]$	$[]$	
$[(XY)^{\frac{r}{2}-1} Xy^k]$	$[yx]$	$[(XY)^{\frac{r}{2}-1} Xy^{k+1}]$	$[X]$	$[]$	$1 \leqslant k < \frac{q-3}{2}$
$[(XY)^{\frac{r}{2}-1} Xy^{\frac{q-3}{2}}]$	$[yx]$	$[]$	$[(XY)^{\frac{r}{2}-1} Xy^{\frac{q-3}{2}} X]$	$[]$	
$[(XY)^{\frac{r}{2}-1} Xy^{\frac{q-3}{2}} (XY)^k]$	$[x]$	$[]$	$[(XY)^{\frac{r}{2}-1} Xy^{\frac{q-3}{2}} (XY)^k X]$	$[YY]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(XY)^{\frac{r}{2}-1} Xy^{\frac{q-3}{2}} (XY)^k X]$	$[]$	$[y]$	$[XX]$	$[(XY)^{\frac{r}{2}-1} Xy^{\frac{q-3}{2}} (XY)^{k+1}]$	
$[(XY)^{\frac{r}{2}-1} Xy^{\frac{q-3}{2}} (XY)^{\frac{r}{2}-1} X]$	$[]$	$[yy]$	$[XX]$	$[]$	
$[(YX)^k]$	$[]$	$[y]$	$[XX]$	$[(YX)^k Y]$	$1 \leqslant k < \frac{r}{2}$
$[(YX)^k Y]$	$[x]$	$[]$	$[(YX)^{k+1}]$	$[YY]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(YX)^{\frac{r}{2}-1} Y]$	$[(YX)^{\frac{r}{2}-1} Yx]$	$[]$	$[]$	$[YY]$	
$[(YX)^{\frac{r}{2}-1} Yx^k]$	$[(YX)^{\frac{r}{2}-1} Yx^{k+1}]$	$[xy]$	$[]$	$[Y]$	$1 \leqslant k < \frac{p-1}{2}$
$[(YX)^{\frac{r}{2}-1} Yx^{\frac{p-1}{2}}]$	$[]$	$[xy]$	$[]$	$[(YX)^{\frac{r}{2}-1} Yx^{\frac{p-1}{2}} Y]$	
$[(YX)^{\frac{r}{2}-1} Yx^{\frac{p-1}{2}} (YX)^k]$	$[]$	$[y]$	$[XX]$	$[(YX)^{\frac{r}{2}-1} Yx^{\frac{p-1}{2}} (YX)^k Y]$	$1 \leqslant k < \frac{r}{2} - 1$
$[(YX)^{\frac{r}{2}-1} Yx^{\frac{p-1}{2}} (YX)^k Y]$	$[x]$	$[]$	$[(YX)^{\frac{r}{2}-1} Yx^{\frac{p-1}{2}} (YX)^{k+1}]$	$[YY]$	$0 \leqslant k < \frac{r}{2} - 1$
$[(YX)^{\frac{r}{2}-1} Yx^{\frac{p-1}{2}} (YX)^{\frac{r}{2}-1} Y]$	$[xx]$	$[]$	$[]$	$[YY]$	

C.6 $p \equiv_2 1$, $q \equiv_2 0$, $r \equiv_2 1$

1	xX	$\rightarrow$	ε
2	Xx	$\rightarrow$	ε
3	yY	$\rightarrow$	ε
4	Yy	$\rightarrow$	ε
5	$x^{\frac{p+1}{2}}$	$\rightarrow$	$X^{\frac{p-1}{2}}$
6	$X^{\frac{p+1}{2}}$	$\rightarrow$	$x^{\frac{p-1}{2}}$
7	$y^{\frac{q}{2}+1}$	$\rightarrow$	$Y^{\frac{q}{2}-1}$
8	$Y^{\frac{q}{2}}$	$\rightarrow$	$y^{\frac{q}{2}}$
9	$(xy)^{\frac{r+1}{2}}$	$\rightarrow$	$(YX)^{\frac{r-1}{2}}$
10	$(yx)^{\frac{r+1}{2}}$	$\rightarrow$	$(XY)^{\frac{r-1}{2}}$
11	$(YX)^{\frac{r-1}{2}}Y$	$\rightarrow$	$(xy)^{\frac{r-1}{2}}x$
12	$(XY)^{\frac{r-1}{2}}X$	$\rightarrow$	$(yx)^{\frac{r-1}{2}}y$
13	$X^{\frac{p-1}{2}}y(xy)^{\frac{r-1}{2}}$	$\rightarrow$	$x^{\frac{p-1}{2}}(YX)^{\frac{r-1}{2}}$
14	$(yx)^{\frac{r-1}{2}}yX^{\frac{p-1}{2}}$	$\rightarrow$	$(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}$
15	$Y^{\frac{q}{2}-1}x(yx)^{\frac{r-1}{2}}$	$\rightarrow$	$y^{\frac{q}{2}}(XY)^{\frac{r-1}{2}}$
16	$(YX)^{\frac{r-1}{2}}y^{\frac{q}{2}}$	$\rightarrow$	$(xy)^{\frac{r-1}{2}}xY^{\frac{q}{2}-1}$
17	$y(xy)^{\frac{r-1}{2}}X^{\frac{p-3}{2}}y(xy)^{\frac{r-1}{2}}$	$\rightarrow$	$(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}(XY)^{\frac{r-1}{2}}$
18	$y^{\frac{q}{2}}(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}$	$\rightarrow$	$Y^{\frac{q}{2}-1}(xy)^{\frac{r-1}{2}}X^{\frac{p-1}{2}}$
19	$(xy)^{\frac{r-1}{2}}xY^{\frac{q}{2}-1}(xy)^{\frac{r-1}{2}}X^{\frac{p-1}{2}}$	$\rightarrow$	$(YX)^{\frac{r-1}{2}}y^{\frac{q}{2}-1}(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}$
20	$(xy)^{\frac{r-1}{2}}xY^{\frac{q}{2}-1}(xy)^{\frac{r-1}{2}}X^{\frac{p-3}{2}}y(xy)^{\frac{r-1}{2}}$	$\rightarrow$	$(YX)^{\frac{r-1}{2}}y^{\frac{q}{2}-1}(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}(YX)^{\frac{r-1}{2}}$
21	$X^{\frac{p-1}{2}}(yx)^{\frac{r-1}{2}}Y^{\frac{q}{2}-1}(xy)^{\frac{r-1}{2}}X^{\frac{p-1}{2}}$	$\rightarrow$	$x^{\frac{p-1}{2}}(YX)^{\frac{r-1}{2}}y^{\frac{q}{2}-1}(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}$
22	$(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}(YX)^{\frac{r-1}{2}}y^{\frac{q}{2}-1}(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}$	$\rightarrow$	$(yx)^{\frac{r-1}{2}}yX^{\frac{p-3}{2}}(yx)^{\frac{r-1}{2}}Y^{\frac{q}{2}-1}(xy)^{\frac{r-1}{2}}X^{\frac{p-1}{2}}$
23	$X^{\frac{p-1}{2}}(yx)^{\frac{r-1}{2}}Y^{\frac{q}{2}-1}(xy)^{\frac{r-1}{2}}X^{\frac{p-3}{2}}y(xy)^{\frac{r-1}{2}}$	$\rightarrow$	$x^{\frac{p-1}{2}}(YX)^{\frac{r-1}{2}}y^{\frac{q}{2}-1}(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}(YX)^{\frac{r-1}{2}}$

State	x	y	X	Y	
$[]$	$[]$	$[]$	$[]$	$[]$	
$[\varepsilon]$	$[x]$	$[y]$	$[X]$	$[Y]$	
$[x^k]$	$[x^{k+1}]$	$[xy]$	$[]$	$[Y]$	$1 \leqslant k < \frac{p-1}{2}$
$[x^{\frac{q-1}{2}}]$	$[]$	$[xy]$	$[]$	$[Y]$	
$[y^k]$	$[yx]$	$[y^{k+1}]$	$[X]$	$[]$	$1 \leqslant k < \frac{q}{2}$
$[y^{\frac{q}{2}}]$	$[yx]$	$[]$	$[y^{\frac{q}{2}}X]$	$[]$	
$[y^{\frac{q}{2}}(XY)^k]$	$[x]$	$[]$	$[y^{\frac{q}{2}}(XY)^kX]$	$[YY]$	$1 \leqslant k < \frac{r-1}{2}$
$[y^{\frac{q}{2}}(XY)^kX]$	$[]$	$[y]$	$[XX]$	$[y^{\frac{q}{2}}(XY)^{k+1}]$	$0 \leqslant k < \frac{r-1}{2}$
$[y^{\frac{q}{2}}(XY)^{\frac{r-1}{2}}]$	$[xx]$	$[]$	$[]$	$[YY]$	
$[X^k]$	$[]$	$[y]$	$[X^{k+1}]$	$[XY]$	$1 \leqslant k < \frac{p-1}{2}$
$[X^{\frac{p-1}{2}}]$	$[]$	$[xy]$	$[]$	$[XY]$	
$[Y^k]$	$[x]$	$[]$	$[YX]$	$[Y^{k+1}]$	$1 \leqslant k < \frac{q}{2} - 1$
$[Y^{\frac{q}{2}-1}]$	$[Y^{\frac{q}{2}-1}x]$	$[]$	$[YX]$	$[]$	
$[Y^{\frac{q}{2}-1}(xy)^k]$	$[Y^{\frac{q}{2}-1}(xy)^kx]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r-3}{2}$
$[Y^{\frac{q}{2}-1}(xy)^kx]$	$[xx]$	$[Y^{\frac{q}{2}-1}(xy)^{k+1}]$	$[]$	$[Y]$	$0 \leqslant k < \frac{r-3}{2}$
$[Y^{\frac{q}{2}-1}(xy)^{\frac{r-1}{2}}]$	$[]$	$[yy]$	$[X]$	$[]$	
$[(xy)^k]$	$[(xy)^kx]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k \leqslant \frac{r-1}{2}$
$[(xy)^kx]$	$[xx]$	$[(xy)^{k+1}]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r-1}{2}$
$[(xy)^{\frac{r-1}{2}}x]$	$[xx]$	$[]$	$[]$	$[(xy)^{\frac{r-1}{2}}xY]$	
$[(xy)^{\frac{r-1}{2}}xY^k]$	$[x]$	$[]$	$[YX]$	$[(xy)^{\frac{r-1}{2}}xY^{k+1}]$	$1 \leqslant k < \frac{q}{2} - 1$
$[(xy)^{\frac{r-1}{2}}xY^{\frac{q}{2}-1}]$	$[yx]$	$[]$	$[YX]$	$[]$	
$[(yx)^k]$	$[xx]$	$[(yx)^ky]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r-1}{2}$
$[(yx)^ky]$	$[(yx)^{k+1}]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r-1}{2}$
$[(yx)^{\frac{r-1}{2}}y]$	$[]$	$[yy]$	$[(yx)^{\frac{r-1}{2}}yX]$	$[]$	
$[(yx)^{\frac{r-1}{2}}yX^k]$	$[]$	$[y]$	$[(yx)^{\frac{r-1}{2}}yX^{k+1}]$	$[XY]$	$1 \leqslant k < \frac{p-3}{2}$
$[(yx)^{\frac{r-1}{2}}yX^{\frac{p-3}{2}}]$	$[]$	$[(yx)^{\frac{r-1}{2}}yX^{\frac{p-3}{2}}y]$	$[]$	$[XY]$	
$[(yx)^{\frac{r-1}{2}}yX^{\frac{p-3}{2}}(yx)^k]$	$[xx]$	$[(yx)^{\frac{r-1}{2}}yX^{\frac{p-3}{2}}(yx)^ky]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r-1}{2}$
$[(yx)^{\frac{r-1}{2}}yX^{\frac{p-3}{2}}(yx)^ky]$	$[(yx)^{\frac{r-1}{2}}yX^{\frac{p-3}{2}}(yx)^{k+1}]$	$[yy]$	$[X]$	$[]$	$0 \leqslant k < \frac{r-1}{2}$
$[(yx)^{\frac{r-1}{2}}yX^{\frac{p-3}{2}}(yx)^{\frac{r-1}{2}}]$	$[xx]$	$[]$	$[]$	$[Y]$	
$[(XY)^k]$	$[x]$	$[]$	$[(XY)^kX]$	$[YY]$	$1 \leqslant k < \frac{r-1}{2}$
$[(XY)^kX]$	$[]$	$[y]$	$[XX]$	$[(XY)^{k+1}]$	$1 \leqslant k < \frac{r-1}{2}$
$[(XY)^{\frac{r-1}{2}}]$	$[(XY)^{\frac{r-1}{2}}x]$	$[]$	$[]$	$[YY]$	
$[(XY)^{\frac{r-1}{2}}x^k]$	$[(XY)^{\frac{r-1}{2}}x^{k+1}]$	$[xy]$	$[]$	$[Y]$	$1 \leqslant k < \frac{p-1}{2}$
$[(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}]$	$[]$	$[xy]$	$[]$	$[(XY)^{\frac{r-1}{2}}x^{\frac{p}{2}-1}]Y$	
$[(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}(YX)^kY]$	$[x]$	$[]$	$[(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}(YX)^{k+1}]$	$[YY]Y$	$0 \leqslant k < \frac{r-1}{2}$
$[(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}(YX)^k]$	$[]$	$[y]$	$[XX]$	$[(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}(YX)^kY]$	$0 \leqslant k < \frac{r-1}{2}$
$[(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}(YX)^{\frac{r-1}{2}}]$	$[]$	$[yy]$	$[XX]$	$[]$	
$[(YX)^k]$	$[]$	$[y]$	$[XX]$	$[(YX)^kY]$	$1 \leqslant k < \frac{r-1}{2}$
$[(YX)^kY]$	$[x]$	$[]$	$[(YX)^{k+1}]$	$[YY]$	$1 \leqslant k < \frac{r-1}{2}$
$[(YX)^{\frac{r-1}{2}}]$	$[]$	$[(YX)^{\frac{r-1}{2}}y]$	$[XX]$	$[]$	
$[(YX)^{\frac{r-1}{2}}y^k]$	$[yx]$	$[(YX)^{\frac{r-1}{2}}y^{k+1}]$	$[X]$	$[]$	$1 \leqslant k < \frac{q}{2} - 1$
$[(YX)^{\frac{r-1}{2}}y^{\frac{q}{2}-1}]$	$[yx]$	$[]$	$[X]$	$[]$	

C.7 $p \equiv_2 0$, $q \equiv_2 1$, $r \equiv_2 1$

1	xX	$\rightarrow$	ε
2	Xx	$\rightarrow$	ε
3	yY	$\rightarrow$	ε
4	Yy	$\rightarrow$	ε
5	$x^{\frac{p}{2}+1}$	$\rightarrow$	$X^{\frac{p}{2}-1}$
6	$X^{\frac{p}{2}}$	$\rightarrow$	$x^{\frac{p}{2}}$
7	$y^{\frac{q+1}{2}}$	$\rightarrow$	$Y^{\frac{q-1}{2}}$
8	$Y^{\frac{q+1}{2}}$	$\rightarrow$	$y^{\frac{q-1}{2}}$
9	$(xy)^{\frac{r+1}{2}}$	$\rightarrow$	$(YX)^{\frac{r-1}{2}}$
10	$(yx)^{\frac{r+1}{2}}$	$\rightarrow$	$(XY)^{\frac{r-1}{2}}$
11	$(YX)^{\frac{r-1}{2}}Y$	$\rightarrow$	$(xy)^{\frac{r-1}{2}}x$
12	$(XY)^{\frac{r-1}{2}}X$	$\rightarrow$	$(yx)^{\frac{r-1}{2}}y$
13	$X^{\frac{p}{2}-1}y(xy)^{\frac{r-1}{2}}$	$\rightarrow$	$x^{\frac{p}{2}}(YX)^{\frac{r-1}{2}}$
14	$(XY)^{\frac{r-1}{2}}x^{\frac{p}{2}}$	$\rightarrow$	$(yx)^{\frac{r-1}{2}}yX^{\frac{p}{2}-1}$
15	$Y^{\frac{q-1}{2}}x(yx)^{\frac{r-1}{2}}$	$\rightarrow$	$y^{\frac{q-1}{2}}(XY)^{\frac{r-1}{2}}$
16	$(xy)^{\frac{r-1}{2}}xY^{\lfloor\frac{q-1}{2}\rfloor}$	$\rightarrow$	$(YX)^{\frac{r-1}{2}}y^{\lceil\frac{q-1}{2}\rceil}$
17	$x(yx)^{\frac{r-1}{2}}Y^{\frac{q-3}{2}}x(yx)^{\frac{r-1}{2}}$	$\rightarrow$	$(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}(YX)^{\frac{r-1}{2}}$
18	$x^{\frac{p}{2}}(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}$	$\rightarrow$	$X^{\frac{p}{2}-1}(yx)^{\frac{r-1}{2}}Y^{\frac{q-1}{2}}$
19	$(yx)^{\frac{r-1}{2}}yX^{\frac{p}{2}-1}(yx)^{\frac{r-1}{2}}Y^{\frac{q-1}{2}}$	$\rightarrow$	$(XY)^{\frac{r-1}{2}}x^{\frac{p}{2}-1}(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}$
20	$(yx)^{\frac{r-1}{2}}yX^{\frac{p}{2}-1}(yx)^{\frac{r-1}{2}}Y^{\frac{q-3}{2}}x(yx)^{\frac{r-1}{2}}$	$\rightarrow$	$(XY)^{\frac{r-1}{2}}x^{\frac{p}{2}-1}(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}(XY)^{\frac{r-1}{2}}$
21	$Y^{\frac{q-1}{2}}(xy)^{\frac{r-1}{2}}X^{\frac{p}{2}-1}(yx)^{\frac{r-1}{2}}Y^{\frac{q-1}{2}}$	$\rightarrow$	$y^{\frac{q-1}{2}}(XY)\frac{r-1}{2}x^{\frac{p}{2}-1}(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}$
22	$(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}(XY)^{\frac{r-1}{2}}x^{\frac{p}{2}-1}(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}$	$\rightarrow$	$(xy)^{\frac{r-1}{2}}xY^{\frac{q-3}{2}}(xy)^{\frac{r-1}{2}}X^{\frac{p}{2}-1}(yx)^{\frac{r-1}{2}}Y^{\frac{q-1}{2}}$
23	$Y^{\frac{q-1}{2}}(xy)^{\frac{r-1}{2}}X^{\frac{p}{2}-1}(yx)^{\frac{r-1}{2}}Y^{\frac{q-3}{2}}x(yx)^{\frac{r-1}{2}}$	$\rightarrow$	$y^{\frac{q-1}{2}}(XY)\frac{r-1}{2}x^{\frac{p}{2}-1}(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}(XY)^{\frac{r-1}{2}}$

State	x	y	X	Y	
$[]$	$[]$	$[]$	$[]$	$[]$	
$[\varepsilon]$	$[x]$	$[y]$	$[X]$	$[Y]$	
$[x^k]$	$[x^{k+1}]$	$[xy]$	$[]$	$[Y]$	$1 \leqslant k < \frac{p}{2}$
$[x^{\frac{p}{2}}]$	$[]$	$[xy]$	$[]$	$[x^{\frac{p}{2}}Y]$	
$[x^{\frac{p}{2}}(YX)^kY]$	$[x]$	$[]$	$[x^{\frac{p}{2}}(YX)^{k+1}]$	$[YY]$	$0 \leqslant k < \frac{r-1}{2}$
$[x^{\frac{p}{2}}(YX)^k]$	$[]$	$[y]$	$[XX]$	$[x^{\frac{p}{2}}(YX)^kY]$	$1 \leqslant k < \frac{r-1}{2}$
$[x^{\frac{p}{2}}(YX)^{\frac{r-1}{2}}]$	$[]$	$[yy]$	$[XX]$	$[]$	
$[y^k]$	$[yx]$	$[y^{k+1}]$	$[X]$	$[]$	$1 \leqslant k < \frac{q-1}{2}$
$[y^{\frac{q-1}{2}}]$	$[yx]$	$[]$	$[X]$	$[]$	
$[X^k]$	$[]$	$[y]$	$[X^{k+1}]$	$[XY]$	$1 \leqslant k < \frac{p}{2}-1$
$[X^{\frac{p}{2}-1}]$	$[]$	$[X^{\frac{p}{2}-1}y]$	$[]$	$[Y]$	
$[X^{\frac{p}{2}-1}(yx)^ky]$	$[X^{\frac{p}{2}-1}(yx)^{k+1}]$	$[yy]$	$[X]$	$[]$	$0 \leqslant k < \frac{r-1}{2}$
$[X^{\frac{p}{2}-1}(yx)^k]$	$[xx]$	$[X^{\frac{p}{2}-1}(yx)^ky]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r-1}{2}$
$[X^{\frac{p}{2}-1}(yx)^{\frac{r-1}{2}}]$	$[xx]$	$[]$	$[]$	$[Y]$	
$[Y^k]$	$[x]$	$[]$	$[YX]$	$[Y^{k+1}]$	$1 \leqslant k < \frac{q-1}{2}$
$[Y^{\frac{q-1}{2}}]$	$[yx]$	$[]$	$[YX]$	$[]$	
$[(xy)^k]$	$[(xy)^kx]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r-1}{2}$
$[(xy)^kx]$	$[xx]$	$[(xy)^{k+1}]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r-1}{2}$
$[(xy)^{\frac{r-1}{2}}x]$	$[xx]$	$[]$	$[]$	$[(xy)^{\frac{r-1}{2}}xY]$	
$[(xy)^{\frac{r-1}{2}}xY^k]$	$[x]$	$[]$	$[YX]$	$[(xy)^{\frac{r-1}{2}}xY^{k+1}]$	$1 \leqslant k < \frac{q-1}{2}$
$[(xy)^{\frac{r-1}{2}}xY^{\frac{q-3}{2}}]$	$[(xy)^{\frac{r-1}{2}}xY^{\frac{q-3}{2}}x]$	$[]$	$[YX]$	$[]$	
$[(xy)^{\frac{r-1}{2}}xY^{\frac{q-3}{2}}(xy)^k]$	$[(xy)^{\frac{r-1}{2}}xY^{\frac{q-3}{2}}(xy)^kx]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r-1}{2}$
$[(xy)^{\frac{r-1}{2}}xY^{\frac{q-3}{2}}(xy)^kx]$	$[xx]$	$[(xy)^{\frac{r-1}{2}}xY^{\frac{q-3}{2}}(xy)^{k+1}]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r-1}{2}$
$[(xy)^{\frac{r-1}{2}}xY^{\frac{q-3}{2}}(xy)^{\frac{r-1}{2}}]$	$[]$	$[yy]$	$[X]$	$[]$	
$[(yx)^k]$	$[xx]$	$[(yx)^ky]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r-1}{2}$
$[(yx)^ky]$	$[(yx)^{k+1}]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r-1}{2}$
$[(yx)^{\frac{r-1}{2}}y]$	$[]$	$[yy]$	$[(yx)^{\frac{r-1}{2}}yX]$	$[]$	
$[(yx)^{\frac{r-1}{2}}yX^k]$	$[]$	$[y]$	$[(yx)^{\frac{r-1}{2}}yX^{k+1}]$	$[XY]$	$1 \leqslant k < \frac{p}{2}-1$
$[(yx)^{\frac{r-1}{2}}yX^{\frac{p}{2}-1}]$	$[]$	$[xy]$	$[]$	$[XY]$	
$[(XY)^k]$	$[x]$	$[]$	$[(XY)^kX]$	$[YY]$	$1 \leqslant k < \frac{r-1}{2}$
$[(XY)^kX]$	$[]$	$[y]$	$[XX]$	$[(XY)^{k+1}]$	$1 \leqslant k < \frac{r-1}{2}$
$[(XY)^{\frac{r-1}{2}}]$	$[(XY)^{\frac{r-1}{2}}x]$	$[]$	$[]$	$[YY]$	
$[(XY)^{\frac{r-1}{2}}x^k]$	$[(XY)^{\frac{r-1}{2}}x^{k+1}]$	$[xy]$	$[]$	$[Y]$	$1 \leqslant k < \frac{p}{2}-1$
$[(XY)^{\frac{r-1}{2}}x^{\frac{p}{2}-1}]$	$[]$	$[xy]$	$[]$	$[Y]$	
$[(YX)^k]$	$[]$	$[y]$	$[XX]$	$[(YX)^kY]$	$1 \leqslant k < \frac{r-1}{2}$
$[(YX)^kY]$	$[x]$	$[]$	$[YX]^{k+1}$	$[YY]$	$1 \leqslant k < \frac{r-1}{2}$
$[(YX)^{\frac{r-1}{2}}]$	$[]$	$[(YX)^{\frac{r-1}{2}}y]$	$[XX]$	$[]$	
$[(YX)^{\frac{r-1}{2}}y^k]$	$[yx]$	$[(YX)^{\frac{r-1}{2}}y^{k+1}]$	$[X]$	$[]$	$1 \leqslant k < \frac{q-1}{2}$
$[(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}]$	$[yx]$	$[]$	$[(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}X]$	$[]$	
$[(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}(XY)^k]$	$[x]$	$[]$	$[(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}(XY)^kX]$	$[YY]$	$1 \leqslant k < \frac{r-1}{2}$
$[(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}(XY)^kX]$	$[]$	$[y]$	$[XX]$	$[(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}(XY)^{k+1}]$	$0 \leqslant k < \frac{r-1}{2}$
$[(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}(XY)^{\frac{r-1}{2}}]$	$[xx]$	$[]$	$[]$	$[YY]$	

C.8 $p \equiv_2 1$, $q \equiv_2 1$, $r \equiv_2 1$

With the exception of the $(3,3,3)$-triangle group.

1	xX	$\rightarrow$	ε
2	Xx	$\rightarrow$	ε
3	yY	$\rightarrow$	ε
4	Yy	$\rightarrow$	ε
5	$x^{\frac{p+1}{2}}$	$\rightarrow$	$X^{\frac{p-1}{2}}$
6	$X^{\frac{p+1}{2}}$	$\rightarrow$	$x^{\frac{p-1}{2}}$
7	$y^{\frac{q+1}{2}}$	$\rightarrow$	$Y^{\frac{q-1}{2}}$
8	$Y^{\frac{q+1}{2}}$	$\rightarrow$	$y^{\frac{q-1}{2}}$
9	$(xy)^{\frac{r+1}{2}}$	$\rightarrow$	$(YX)^{\frac{r-1}{2}}$
10	$(yx)^{\frac{r+1}{2}}$	$\rightarrow$	$(XY)^{\frac{r-1}{2}}$
11	$(YX)^{\frac{r-1}{2}}Y$	$\rightarrow$	$(xy)^{\frac{r-1}{2}}x$
12	$(XY)^{\frac{r-1}{2}}X$	$\rightarrow$	$(yx)^{\frac{r-1}{2}}y$
13	$X^{\frac{p-1}{2}}y(xy)\,\frac{r-1}{2}$	$\rightarrow$	$x^{\frac{p-1}{2}}(YX)^{\frac{r-1}{2}}$
14	$(yx)^{\frac{r-1}{2}}yX^{\frac{p-1}{2}}$	$\rightarrow$	$(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}$
15	$Y^{\frac{q-1}{2}}x(yx)^{\frac{r-1}{2}}$	$\rightarrow$	$y^{\frac{q-1}{2}}(XY)^{\frac{r-1}{2}}$
16	$(xy)^{\frac{r-1}{2}}xY^{\frac{q-1}{2}}$	$\rightarrow$	$(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}$
17	$X^{\frac{p-1}{2}}(yx)^{\frac{r-1}{2}}Y^{\frac{q-1}{2}}$	$\rightarrow$	$x^{\frac{p-1}{2}}(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}$
18	$Y^{\frac{q-1}{2}}(xy)^{\frac{r-1}{2}}X^{\frac{p-1}{2}}$	$\rightarrow$	$y^{\frac{q-1}{2}}(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}$
19	$y(xy)^{\frac{r-1}{2}}X^{\frac{p-3}{2}}y(xy)^{\frac{r-1}{2}}$	$\rightarrow$	$(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}(YX)^{\frac{r-1}{2}}$
20	$x(yx)^{\frac{r-1}{2}}Y^{\frac{q-3}{2}}x(yx)^{\frac{r-1}{2}}$	$\rightarrow$	$(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}(XY)^{\frac{r-1}{2}}$
21	$X^{\frac{p-1}{2}}(yx)^{\frac{r-1}{2}}Y^{\frac{q-3}{2}}(xy)^{\frac{r-1}{2}}x$	$\rightarrow$	$x^{\frac{p-1}{2}}(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}(XY)^{\frac{r-1}{2}}$
22	$(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}$	$\rightarrow$	$(yx)^{\frac{r-1}{2}}yX^{\frac{p-3}{2}}(yx)^{\frac{r-1}{2}}Y^{\frac{q-1}{2}}$
23	$Y^{\frac{q-1}{2}}(xy)^{\frac{r-1}{2}}X^{\frac{p-3}{2}}(yx)^{\frac{r-1}{2}}y$	$\rightarrow$	$y^{\frac{q-1}{2}}(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}(YX)^{\frac{r-1}{2}}$
24	$(YX)^{\frac{r-1}{2}}y^{\frac{q-1}{2}}(XY)^{\frac{r-1}{2}}x^{\frac{p-1}{2}}$	$\rightarrow$	$(xy)^{\frac{r-1}{2}}xY^{\frac{q-3}{2}}(xy)^{\frac{r-1}{2}}X^{\frac{p-1}{2}}$

State	x	y	X	Y	
$[]$	$[]$	$[]$	$[]$	$[]$	
$[\varepsilon]$	$[x]$	$[y]$	$[X]$	$[Y]$	
$[x^k]$	$[x^{k+1}]$	$[xy]$	$[]$	$[Y]$	$1 \leqslant k < \frac{p-1}{2}$
$[x^{\frac{p-1}{2}}]$	$[]$	$[xy]$	$[]$	$[Y]$	
$[y^k]$	$[yx]$	$[y^{k+1}]$	$[X]$	$[]$	$1 \leqslant k < \frac{q-1}{2}$
$[y^{\frac{q-1}{2}}]$	$[yx]$	$[]$	$[X]$	$[]$	
$[X^k]$	$[]$	$[y]$	$[X^{k+1}]$	$[XY]$	$1 \leqslant k < \frac{p-1}{2}$
$[X^{\frac{p-1}{2}}]$	$[]$	$[xy]$	$[]$	$[XY]$	
$[Y^k]$	$[x]$	$[]$	$[YX]$	$[Y^{k+1}]$	$1 \leqslant k < \frac{q-1}{2}$
$[Y^{\frac{q-1}{2}}]$	$[yx]$	$[]$	$[YX]$	$[]$	
$[(xy)^k]$	$[(xy)^k x]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r-1}{2}$
$[(xy)^k x]$	$[xx]$	$[(xy)^{k+1}]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r-1}{2}$
$[(xy)^{\frac{r-1}{2}} x]$	$[xx]$	$[]$	$[]$	$[(xy)^{\frac{r-1}{2}} xY]$	
$[(xy)^{\frac{r-1}{2}} xY^k]$	$[x]$	$[]$	$[YX]$	$[(xy)^{\frac{r-1}{2}} xY^{k+1}]$	$1 \leqslant k < \frac{q-3}{2}$
$[(xy)^{\frac{r-1}{2}} xY^{\frac{q-3}{2}}]$	$[(xy)^{\frac{r-1}{2}} xY^{\frac{q-3}{2}} x]$	$[]$	$[YX]$	$[]$	
$[(xy)^{\frac{r-1}{2}} xY^{\frac{q-3}{2}} (xy)^k x]$	$[xx]$	$[(xy)^{\frac{r-1}{2}} xY^{\frac{q-3}{2}} (xy)^{k+1}]$	$[]$	$[Y]$	$0 \leqslant k < \frac{r-1}{2}$
$[(xy)^{\frac{r-1}{2}} xY^{\frac{q-3}{2}} (xy)^k]$	$[(xy)^{\frac{r-1}{2}} xY^{\frac{q-3}{2}} (xy)^k x]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r-1}{2}$
$[(xy)^{\frac{r-1}{2}} xY^{\frac{q-3}{2}} (xy)^{\frac{r-1}{2}}]$	$[]$	$[yy]$	$[X]$	$[]$	
$[(yx)^k]$	$[xx]$	$[(yx)^k y]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r-1}{2}$
$[(yx)^k y]$	$[(yx)^{k+1}]$	$[yy]$	$[X]$	$[]$	$1 \leqslant k < \frac{r-1}{2}$
$[(yx)^{\frac{r-1}{2}} y]$	$[]$	$[yy]$	$[(yx)^{\frac{r-1}{2}} yX]$	$[]$	
$[(yx)^{\frac{r-1}{2}} yX^k]$	$[]$	$[y]$	$[(yx)^{\frac{r-1}{2}} yX^{k+1}]$	$[XY]$	$1 \leqslant k < \frac{p-3}{2}$
$[(yx)^{\frac{r-1}{2}} yX^{\frac{p-3}{2}}]$	$[]$	$[(yx)^{\frac{r-1}{2}} yX^{\frac{p-3}{2}} y]$	$[]$	$[XY]$	
$[(yx)^{\frac{r-1}{2}} yX^{\frac{p-3}{2}} (yx)^k y]$	$[(yx)^{\frac{r-1}{2}} yX^{\frac{p-3}{2}} (yx)^{k+1}]$	$[yy]$	$[X]$	$[]$	$0 \leqslant k < \frac{r-1}{2}$
$[(yx)^{\frac{r-1}{2}} yX^{\frac{p-3}{2}} (yx)^k]$	$[xx]$	$[(yx)^{\frac{r-1}{2}} yX^{\frac{p-3}{2}} (yx)^k y]$	$[]$	$[Y]$	$1 \leqslant k < \frac{r-1}{2}$
$[(yx)^{\frac{r-1}{2}} yX^{\frac{p-3}{2}} (yx)^{\frac{r-1}{2}}]$	$[xx]$	$[]$	$[]$	$[Y]$	
$[(XY)^k]$	$[x]$	$[]$	$[(XY)^k X]$	$[YY]$	$1 \leqslant k < \frac{r-1}{2}$
$[(XY)^k X]$	$[]$	$[y]$	$[XX]$	$[(XY)^{k+1}]$	$1 \leqslant k < \frac{r-1}{2}$
$[(XY)^{\frac{r-1}{2}}]$	$[(XY)^{\frac{r-1}{2}} x]$	$[]$	$[]$	$[YY]$	
$[(XY)^{\frac{r-1}{2}} x^k]$	$[(XY)^{\frac{r-1}{2}} x^{k+1}]$	$[xy]$	$[]$	$[Y]$	$1 \leqslant k < \frac{p-1}{2}$
$[(XY)^{\frac{r-1}{2}} x^{\frac{p-1}{2}}]$	$[]$	$[xy]$	$[]$	$[(XY)^{\frac{r-1}{2}} x^{\frac{p-1}{2}} Y]$	
$[(XY)^{\frac{r-1}{2}} x^{\frac{p-1}{2}} (YX)^k]$	$[]$	$[y]$	$[XX]$	$[(XY)^{\frac{r-1}{2}} x^{\frac{p-1}{2}} (YX)^k Y]$	$0 \leqslant k < \frac{r-1}{2}$
$[(XY)^{\frac{r-1}{2}} x^{\frac{p-1}{2}} (YX)^k Y]$	$[x]$	$[]$	$[(XY)^{\frac{r-1}{2}} x^{\frac{p-1}{2}} (YX)^{k+1}]$	$[YY]$	$0 \leqslant k < \frac{r-1}{2}$
$[(XY)^{\frac{r-1}{2}} x^{\frac{p-1}{2}} (YX)^{\frac{r-1}{2}}]$	$[]$	$[yy]$	$[XX]$	$[]$	
$[(YX)^k]$	$[]$	$[y]$	$[XX]$	$[(YX)^k Y]$	$1 \leqslant k < \frac{r-1}{2}$
$[(YX)^k Y]$	$[x]$	$[]$	$[(YX)^{k+1}]$	$[YY]$	$1 \leqslant k < \frac{r-1}{2}$
$[(YX)^{\frac{r-1}{2}}]$	$[]$	$[(YX)^{\frac{r-1}{2}} y]$	$[XX]$	$[]$	
$[(YX)^{\frac{r-1}{2}} y^k]$	$[yx]$	$[(YX)^{\frac{r-1}{2}} y^{k+1}]$	$[X]$	$[]$	$1 \leqslant k < \frac{q-1}{2}$
$[(YX)^{\frac{r-1}{2}} y^{\frac{q-1}{2}}]$	$[yx]$	$[]$	$[(YX)^{\frac{r-1}{2}} y^{\frac{q-1}{2}}]X$	$[]$	
$[(YX)^{\frac{r-1}{2}} y^{\frac{q-1}{2}} (XY)^k]$	$[x]$	$[]$	$[(YX)^{\frac{r-1}{2}} y^{\frac{q-3}{2}} (XY)^k X]$	$[YY]$	$1 \leqslant k < \frac{r-1}{2}$
$[(YX)^{\frac{r-1}{2}} y^{\frac{q-1}{2}} (XY)^k X]$	$[]$	$[y]$	$[XX]$	$[(YX)^{\frac{r-1}{2}} y^{\frac{q-1}{2}} (XY)^{k+1}]$	$0 \leqslant k < \frac{r-1}{2}$
$[(YX)^{\frac{r-1}{2}} y^{\frac{q-3}{2}} (XY)^{\frac{r-1}{2}}]$	$[xx]$	$[]$	$[]$	$[YY]$	

D Growth Functions

This section lists a few examples of growth functions of triangle groups for some selected parameters.

p	q	r	Growth Function
6	6	6	$\frac{1+2z+2z^2+2z^3+2z^4+2z^5+z^6}{1-2z-2z^2-2z^4-2z^5+z^6}$
8	6	6	$\frac{1+2z+3z^2+4z^3+4z^4+4z^5+3z^6+2z^7+z^8}{1-2z-z^2-3z^3-3z^4-3z^5-z^6-2z^7+z^8}$
8	8	6	$\frac{1+2z+3z^2+4z^3+4z^4+4z^5+3z^6+2z^7+z^8}{1-2z-z^2-4z^3-2z^4-4z^5-z^6-2z^7+z^8}$
8	8	8	$\frac{1+2z+2z^2+2z^3+2z^4+2z^5+2z^6+2z^7+z^8}{1-2z-2z^2-2z^3-2z^5-2z^6-2z^7+z^8}$
10	6	6	$\frac{1+3z+5z^2+7z^3+9z^4+10z^5+9z^6+7z^7+5z^8+3z^9+z^{10}}{1-z-3z^2-4z^3-6z^4-8z^5-6z^6-4z^7-3z^8-z^9+z^{10}}$
10	8	6	$\frac{1+3z+6z^2+10z^3+14z^4+17z^5+18z^6+17z^7+14z^8+10z^9+6z^{10}+3z^{11}+z^{12}}{1-z-2z^2-6z^3-9z^4-13z^5-13z^6-13z^7-9z^8-6z^9-2z^{10}-z^{11}+z^{12}}$
10	10	6	$\frac{1+3z+5z^2+7z^3+9z^4+10z^5+9z^6+7z^7+5z^8+3z^9+z^{10}}{1-z-3z^2-5z^3-7z^4-8z^5-7z^6-5z^7-3z^8-z^9+z^{10}}$
10	8	8	$\frac{1+3z+5z^2+7z^3+9z^4+10z^5+10z^6+10z^7+9z^8+7z^9+5z^{10}+3z^{11}+z^{12}}{1-z-3z^2-5z^3-6z^4-8z^5-8z^6-8z^7-6z^8-5z^9-3z^{10}-z^{11}+z^{12}}$
10	10	8	$\frac{1+3z+5z^2+7z^3+9z^4+10z^5+10z^6+10z^7+9z^8+7z^9+5z^{10}+3z^{11}+z^{12}}{1-z-3z^2-5z^3-7z^4-8z^5-8z^6-8z^7-7z^8-5z^9-3z^{10}-z^{11}+z^{12}}$
10	10	10	$\frac{1+2z+2z^2+2z^3+2z^4+2z^5+2z^6+2z^7+2z^8+2z^9+z^{10}}{1-2z-2z^2-2z^3-2z^4-2z^6-2z^7-2z^8-2z^9+z^{10}}$
12	6	6	$\frac{1+2z+2z^2+2z^3+2z^4+2z^5+z^6}{1-2z-2z^2-z^3-2z^4-2z^5+z^6}$
12	8	6	$\frac{1+2z+3z^2+4z^3+4z^4+4z^5+3z^6+2z^7+z^8}{1-2z-z^2-4z^3-3z^4-4z^5-z^6-2z^7+z^8}$
12	10	6	$\frac{1+3z+5z^2+7z^3+9z^4+10z^5+9z^6+7z^7+5z^8+3z^9+z^{10}}{1-z-3z^2-5z^3-7z^4-9z^5-7z^6-5z^7-3z^8-z^9+z^{10}}$
12	12	6	$\frac{1+2z+2z^2+2z^3+2z^4+2z^5+z^6}{1-2z-2z^2-2z^3-2z^4-2z^5+z^6}$
12	8	8	$\frac{1+2z+3z^2+4z^3+5z^4+6z^5+6z^6+6z^7+5z^8+4z^9+3z^{10}+2z^{11}+z^{12}}{1-2z-z^2-4z^3-2z^4-6z^5-4z^6-6z^7-2z^8-4z^9-z^{10}-2z^{11}+z^{12}}$
12	10	8	$\frac{1+3z+6z^2+10z^3+15z^4+20z^5+24z^6+27z^7+28z^8+27z^9+24z^{10}+20z^{11}+15z^{12}+10z^{13}+6z^{14}+3z^{15}+z^{16}}{1-z-2z^2-6z^3-9z^4-15z^5-18z^6-22z^7-22z^8-22z^9-18z^{10}-15z^{11}-9z^{12}-6z^{13}-2z^{14}-z^{15}+z^{16}}$
12	12	8	$\frac{1+2z+3z^2+4z^3+5z^4+6z^5+6z^6+6z^7+5z^8+4z^9+3z^{10}+2z^{11}+z^{12}}{1-2z-z^2-4z^3-3z^4-6z^5-4z^6-6z^7-3z^8-4z^9-z^{10}-2z^{11}+z^{12}}$
12	10	10	$\frac{1+2z+3z^2+4z^3+5z^4+6z^5+6z^6+6z^7+6z^8+6z^9+5z^{10}+4z^{11}+3z^{12}+2z^{13}+z^{14}}{1-2z-z^2-4z^3-3z^4-5z^5-5z^6-5z^7-5z^8-5z^9-3z^{10}-4z^{11}-z^{12}-2z^{13}+z^{14}}$
12	12	10	$\frac{1+2z+3z^2+4z^3+5z^4+6z^5+6z^6+6z^7+6z^8+6z^9+5z^{10}+4z^{11}+3z^{12}+2z^{13}+z^{14}}{1-2z-z^2-4z^3-3z^4-6z^5-4z^6-6z^7-4z^8-6z^9-3z^{10}-4z^{11}-z^{12}-2z^{13}+z^{14}}$
12	12	12	$\frac{1+2z+2z^2+2z^3+2z^4+2z^5+2z^6+2z^7+2z^8+2z^9+2z^{10}+2z^{11}+z^{12}}{1-2z-2z^2-2z^3-2z^4-2z^5-2z^7-2z^8-2z^9-2z^{10}-2z^{11}+z^{12}}$
7	6	6	$\frac{1+z+2z^2+2z^3+z^4+3z^5-z^6+3z^7-z^8+3z^9+z^{10}+2z^{11}+2z^{12}+z^{13}+z^{14}}{1-3z+2z^2-5z^3+2z^4-3z^5+z^7-3z^9+2z^{10}-5z^{11}+2z^{12}-3z^{13}+z^{14}}$
9	6	6	$\frac{1+3z+5z^2+7z^3+9z^4+9z^5+8z^6+6z^7+4z^8+4z^9+4z^{10}+6z^{11}+8z^{12}+9z^{13}+9z^{14}+7z^{15}+5z^{16}+3z^{17}+z^{18}}{1-z-3z^2-4z^3-6z^4-8z^5-4z^6-4z^7-2z^8+2z^9-2z^{10}-4z^{11}-4z^{12}-8z^{13}-6z^{14}-4z^{15}-3z^{16}-z^{17}+z^{18}}$
9	8	6	$\frac{1+3z+6z^2+10z^3+14z^4+16z^5+17z^6+15z^7+12z^8+10z^9+8z^{10}+10z^{11}+12z^{12}+15z^{13}+17z^{14}+16z^{15}+14z^{16}+10z^{17}+6z^{18}+3z^{19}+z^{20}}{1-z-2z^2-6z^3-9z^4-13z^5-11z^6-13z^7-5z^8-5z^9-z^{10}-5z^{11}-5z^{12}-13z^{13}-11z^{14}-13z^{15}-9z^{16}-6z^{17}-2z^{18}-z^{19}+z^{20}}$
11	8	6	$\frac{1+5z+13z^2+23z^3+30z^4+30z^5+22z^6+10z^7-4z^9+10z^{11}+22z^{12}+30z^{13}+30z^{14}+23z^{15}+13z^{16}+5z^{17}+z^{18}}{1+z-3z^2-13z^3-25z^4-31z^5-25z^6-7z^7+11z^8+19z^9+11z^{10}-7z^{11}-25z^{12}-31z^{13}-25z^{14}-13z^{15}-3z^{16}+z^{17}+z^{18}}$
7	7	7	$\frac{1+3z+5z^2+7z^3+7z^4+7z^5+7z^6+6z^7+4z^8+4z^9+4z^{10}+6z^{11}+7z^{12}+7z^{13}+7z^{14}+7z^{15}+5z^{16}+3z^{17}+z^{18}}{1-z-3z^2-5z^3-5z^4-5z^5-5z^6-4z^7-2z^8+2z^9-2z^{10}-4z^{11}-5z^{12}-5z^{13}-5z^{14}-5z^{15}-3z^{16}-z^{17}+z^{18}}$
9	9	7	$\frac{1+3z+5z^2+7z^3+9z^4+9z^5+9z^6+8z^7+6z^8+4z^9+4z^{10}+4z^{11}+6z^{12}+8z^{13}+9z^{14}+9z^{15}+9z^{16}+7z^{17}+5z^{18}+3z^{19}+z^{20}}{1-z-3z^2-5z^3-7z^4-7z^5-7z^6-6z^7-4z^8-2z^9+2z^{10}-2z^{11}-4z^{12}-6z^{13}-7z^{14}-7z^{15}-7z^{16}-5z^{17}-3z^{18}-z^{19}+z^{20}}$
9	9	9	$\frac{1+3z+5z^2+7z^3+9z^4+9z^5+9z^6+9z^7+9z^8+8z^9+6z^{10}+4z^{11}+4z^{12}+4z^{13}+6z^{14}+8z^{15}+9z^{16}+9z^{17}+9z^{18}+9z^{19}+9z^{20}+7z^{21}+5z^{22}+3z^{23}+z^{24}}{1-z-3z^2-5z^3-7z^4-7z^5-7z^6-7z^7-7z^8-6z^9-4z^{10}-2z^{11}+2z^{12}-2z^{13}-4z^{14}-6z^{15}-7z^{16}-7z^{17}-7z^{18}-7z^{19}-7z^{20}-5z^{21}-3z^{22}-z^{23}+z^{24}}$

E Transition Matrices

Transition matrix of word acceptor for the $(6,6,6)$-triangle group.

$$
\begin{pmatrix}
1 & . & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & 1 & . & . & . & 1 & . & . & . & . & . & 1 & . & . & . & . & . & . & . & 1 & . & . & 1 & . & . & . & . & . & . & . \\
. & 1 & . & . & 1 & . & . & . & . & . & . & . & 1 & . & . & . & . & . & 1 & . & . & . & . & 1 & . & . & . & . & . & . \\
. & 1 & . & 1 & . & . & . & 1 & . & 1 & . & . & . & . & . & . & 1 & 1 & . & . & 1 & . & . & . & . & 1 & . & . & 1 & . \\
. & 1 & 1 & . & . & . & 1 & . & 1 & . & . & . & . & . & 1 & 1 & . & . & . & . & . & 1 & . & . & 1 & . & . & . & . & 1 \\
. & . & 1 & . & . & . & . & . & 1 & . & . & . & . & . & . & 1 & . & . & . & . & . & 1 & . & . & 1 & . & . & 1 & . & 1 \\
. & . & 1 & . & . & . & 1 & . & . & . & 1 & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & 1 & . & . & . & . & . & 1 & . & . & . & 1 & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & 1 & . & . & . & 1 & . & . & . & . & . & . & . & . & 1 & . & . & . & 1 & . & . & . & . & 1 & 1 & . & 1 & . \\
. & . & . & . & 1 & . & . & . & . & . & . & . & 1 & . & . & . & . & . & 1 & . & . & . & . & 1 & . & . & 1 & . & . & . \\
. & . & . & . & 1 & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & 1 & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & 1 & . & . & . & . & . & 1 & . & . & . & . & . & . & . & 1 & . & . & 1 & . & . & . & . & 1 & . & . \\
. & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & .
\end{pmatrix}
$$

Transition matrix of word acceptor for the $(14,7,6)$-triangle group.

$$
\left(\begin{array}{*{54}{c}}
1 & . & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & 1 & . & . & . & 1 & . & . & . & . & . & 1 & . & 1 & . & . & . & . & . & . & 1 & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & 1 & . & . & . & 1 & . & . \\
. & 1 & . & . & 1 & . & . & . & . & . & 1 & . & 1 & . & . & . & . & . & 1 & 1 & . & . & . & . & . & 1 & . & 1 & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & 1 & . & . & . & . \\
. & 1 & . & 1 & . & . & . & 1 & . & 1 & . & . & . & . & . & . & 1 & 1 & . & . & . & . & . & 1 & . & . & . & . & . & . & 1 & . & . & . & . & 1 & . & . & 1 & . & . & 1 & . & . & . & . & 1 & . & . & . & 1 & . & . & . \\
. & 1 & 1 & . & . & . & 1 & . & 1 & . & . & . & . & . & 1 & 1 & . & . & . & . & . & . & 1 & . & 1 & . & . & . & 1 & 1 & . & . & . & . & 1 & . & . & . & . & 1 & . & . & . & . & 1 & . & . & . & 1 & . & . & . & 1 & . \\
. & . & 1 & . & . & . & . & . & 1 & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & 1 & . & . & . & . & 1 & . & . & . & 1 & . & . & 1 & . & . & . & . & . & . & . & 1 & . & . & . & 1 & . & . & . & 1 & . \\
. & . & 1 & . & . & . & 1 & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & 1 & . & . & . & . & . & 1 & . & . & . & . & . & 1 & . & . & 1 & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & 1 & . & . & . & . & . & 1 & . & . & . & . & . & . & . & 1 & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & 1 & . & 1 & . & . & . & . & . & . & . & . & . & . \\
. & . & . & 1 & . & . & . & 1 & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & 1 & . & . & . & . & . & . & 1 & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & 1 & . & . & 1 \\
. & . & . & . & 1 & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & 1 & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & 1 & . & . & . & 1 \\
. & . & . & . & 1 & . & . & . & . & . & 1 & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & 1 & . & . & . & . & . & 1 & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & 1 & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & 1 & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & 1 & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & 1 & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & 1 & . & . \\
. & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & .
\end{array}\right)
$$

Index

Bibliography

[BG04] Achim Blumensath and Erich Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory of Computing Systems*, 37:641 – 674, 2004.

[Blu99] Achim Blumensath. *Automatic Structures, Diplomarbeit.* 1999.

[Deh11] Max Dehn. Über unendliche diskontinuierliche Gruppen. *Math Ann.*, 71:116–144, 1911.

[dlH00] P. de la Harpe. *Topics in Geometric Group Theory*. The University of Chicago Press, 2000.

[EIFZ96] David B. A. Epstein, Anthony R. Iano-Fletcher, and Uri Zwick. Growth functions and automatic groups. *Experimental mathematics*, 5(4):297–315, 1996.

[EPC$^+$92] David B. A. Epstein, M. S. Paterson, J. W. Cannon, D. F. Holt, S. V. Levy, and W. P. Thurston. *Word Processing in Groups*. A. K. Peters, Ltd., Natick, MA, USA, 1992.

[Gil] Robert H. Gilman. On the definition of word hyperbolic groups.

[HEO05] Derek F. Holt, Bettina Eick, and Eamonn A. O'Brien. *Handbook of Computational Group Theory*. Chapman & Hall/CRC, 2005.

[HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[Hol94] Derek Holt. Knuth-Bendix for Monoids and Groups (kbmag). http://www.warwick.ac.uk/˜mareg/download/kbmag2/, 1994.

[Joh80] D. L. Johnson. *Topics in the Theory of Group Presentations*. Cambridge University Press, 1980.

[KN95] B. Khoussainov and A. Nerode. Automatic presentations of structures. In *LCC '94: Selected Papers from the International Workshop on Logical and Computational Complexity*, volume 960 of *Lecture Notes in Computer Science*, pages 367–392. Springer-Verlag, 1995.

[Nov55] P.S. Novikov. On the Algorithmic Unsolvability of the Word Problem in Group Theory. *Trudy Math. Inst. im. Stevlov*, 44:1–143, 1955.

[OT05] Graham P. Oliver and Richard M. Thomas. Automatic presentations for finitely generated groups. In *STACS*, pages 693–704, 2005.

[Pin00] Jean-Eric Pin. *Syntactic Semigroups*. 2000.

[Ros04] Stefan Rosebrock. *Geometrische Gruppentheorie*. Vieweg, 2004.

[Sim94] Charles C. Sims. *Computation with Finitely Presented Groups*. Cambridge University Press, 1994.